CAPÍTULO 5.
Firebase Functions, Enlaces Dinámicos, Stability y mucho más.

Por VICENTE CARBONELL

En este capítulo vamos a seguir viendo características de *Firebase*. Controlaremos los eventos producidos en los servidores de Firebase mediante el uso de funciones con *Firebase Functions*. Aprenderemos a variar el comportamiento y el aspecto de una aplicación de forma remota con *Firebase Remote Config*. Conoceremos que es y cómo utilizar enlaces dinámicos mediante *Firebase* en *Android* con *Firebase Dynamic Links*. Veremos cómo otorgar estabilidad a una aplicación mediante Firebase. Para ello veremos los servicios: *Crashlitycs* (para capturar errores de la aplicación que se producen en los dispositivos de los usuarios), *Performance* (evaluaremos el rendimiento de la aplicación en distintos puntos para mejorarlo) y *Test Lab* (nos permitirá realizar pruebas de una aplicación en dispositivos reales con distintas configuraciones).

Por último, aprenderemos a usar el servicio de copia de seguridad con *Android Backup Service*. Imaginemos que para que nuestra aplicación funcione le pedimos al usuario una serie de datos o que el funcionamiento de la misma genere datos que vamos guardando de forma local en nuestro dispositivo. Si por cualquier motivo el usuario hiciera un reset de fábrica, cambiara de dispositivo, etc., perdería todos los datos recopilados por nuestra aplicación, siendo muy frustrante para el usuario. Google ofrece un servicio para guardar todos estos datos en la nube de forma transparente, de tal forma que al volver a instalar la aplicación los tendríamos disponibles.

Objetivos:

- Ejecutar funciones en los servidores de Google que son activadas por medio de ciertos eventos.
- Utilizar *Firebase Remote Config* en aplicaciones *Android* para variar su aspecto o comportamiento de forma remota.
- Compartir nuestra aplicación o contenido de la misma utilizando enlaces dinámicos.
- Capturar errores en los dispositivos de los usuarios con *Firebase Crashlitycs*.
- Evaluar el rendimiento de una aplicación con *Firebase Performance*.
- Realizar pruebas de una aplicación en dispositivos reales con *Firebase Test Lab*.
- Crear copias de seguridad en línea usando Android Backup Service.

5.1. Cloud Functions

5.1.1. Introducción

Cloud Functions para *Firebase* permite ejecutar automáticamente código *JavaScript* en los servidores de *Google* en respuesta a eventos en los servidores de *Firebase*. Los eventos son activados por desencadenadores de *Firebase* y solicitudes *HTTP/S*. El código a ejecutar se almacena en la nube de *Google* y se ejecuta en un entorno administrado.

Los distintos tipos de eventos que dispararán los desencadenadores y ejecutarán las funciones de Firebase son eventos de: Cloud Firestore, *Realtime Database*, *Firebase Authentication*, *Google Analytics* para *Firebase*, *Cloud Storage*, *Pub/Sub* de *Cloud* o llamadas *HTTP/S*.

Implementa el código en los servidores de *Firebase* desde una consola en tu ordenador.

Firebase gestiona los recursos de servidor de forma automática según los patrones de uso de los usuarios. No tendrás que preocuparte de las credenciales, la configuración, mantenimiento o aprovisionamiento de servidores.

En muchos casos, los programadores prefieren ejecutar parte de la lógica de la aplicación en un servidor para evitar alteraciones del código en la parte cliente y liberar al dispositivo cliente de consumo de recursos. Así, aunque se aplicara ingeniería inversa a la aplicación, parte del código estaría protegido por estar en el servidor. El código a ejecutar en *Cloud Functions* está completamente aislado del cliente.

Después de implementar una función, los servidores de Google comienzan a

administrarla. Detectan eventos y ejecutan la función cuando se activa. A medida que la carga aumenta o disminuye, la respuesta de Google es escalar con rapidez la cantidad de instancias de servidor virtual necesarias para ejecutar la función.

El ciclo de vida de una función es:

1. Se implementa una nueva función seleccionando un proveedor de eventos y se definen las condiciones según las cuales debe ejecutarse la función.
2. *Firebase* la conecta al proveedor de eventos seleccionado.
3. Cuando el proveedor de eventos genera un evento que coincide con las condiciones de la función, se invoca el código.
4. Si la función está ocupada con muchos eventos, *Google* crea más instancias para controlar el trabajo con más rapidez. Si la función está inactiva, las instancias se borran.
5. Cuando se actualiza una función, todas las instancias de la versión antigua se borran y se reemplazan por instancias nuevas.
6. Cuando un programador borra la función, se borran todas las instancias y se quita la conexión entre la función y el proveedor de eventos.

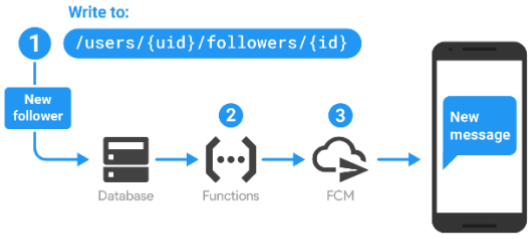
Para utilizar *Firebase Functions* debemos instalar *Firebase CLI* e inicializar *Firebase Functions* en el proyecto de *Firebase*. Posteriormente, podrás escribir el código *JavaScript* de la función. Por último, implementa la función en los servidores de *Firebase* con *Firebase CLI*. Puedes utilizar la consola de *Firebase* para ver y buscar en los registros de funciones implementadas.

Cloud Functions permite que los programadores accedan a eventos de *Firebase* y *Google Cloud* y brinda la capacidad de procesamiento escalable para ejecutar código en respuesta a eventos. Los casos típicos casos donde se usa son:

- Notificar a los usuarios cuando ocurre algo interesante.
- Ejecución de limpieza y mantenimiento en *Realtime Database*.
- Ejecución de tareas intensivas en la nube en lugar de en la aplicación.
- Realizar integraciones con API y servicios de terceros.

Notificar a los usuarios cuando ocurre algo interesante.

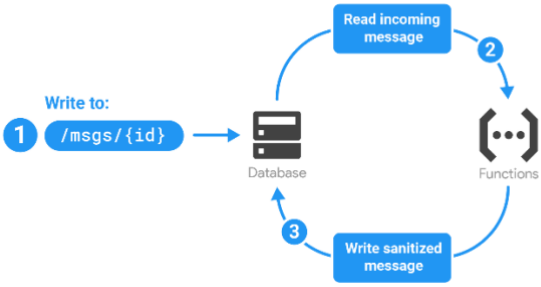
Podemos utilizar *Cloud Functions* para mantener a los usuarios interesados y actualizados con información relevante. Por ejemplo, una función que se activa cuando se escribe en *Realtime Database* para almacenar nuevos registros, podría generar notificaciones de *Firebase Cloud Messaging* (FCM) para informar a los usuarios.



La función se activa cuando se escribe en una ruta de acceso de *Realtime Database* que el programador ha definido. La función crea un mensaje para enviarlo a través de *FCM*. *FCM* envía el mensaje de notificación al dispositivo del usuario.

Ejecución de limpieza y mantenimiento en Realtime Database.

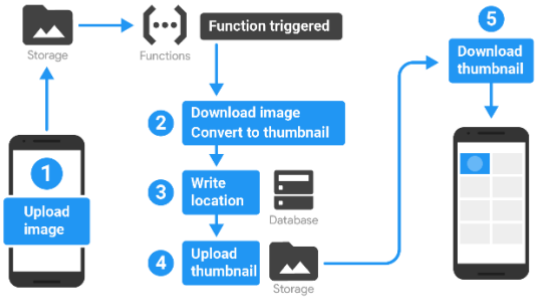
Con *Cloud Functions*, puedes modificar *Realtime Database* para mantener el sistema actualizado y limpio. Por ejemplo, en una aplicación que admita comentarios y que sean almacenados en *Realtime Database*, se podrían supervisar los eventos de escritura y quitar el texto inapropiado.



El controlador de eventos de la base de datos detecta eventos de escritura en una ruta de acceso específica y recupera datos del evento con el texto insertado. La función procesa el texto para detectar y quitar el lenguaje inapropiado. Vuelve a escribir el texto actualizado en la base de datos.

Ejecución de tareas intensivas en la nube en lugar de en la aplicación.

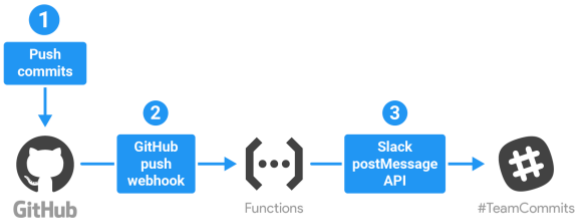
Podemos aprovechar *Cloud Functions* para ejecutar, en la nube de *Google*, trabajo que requiere una gran cantidad de recursos (uso intensivo de CPU o de red) y que no sería práctico ejecutar en el dispositivo de un usuario. Por ejemplo, una función que detecte cuando se suben imágenes a *Firebase Storage*, descargue la imagen a la instancia que ejecuta la función, la modifique y la vuelva a subir a *Storage*. Se podría modificar el tamaño, recortar o convertir la imagen.



La función se activa cuando un archivo de imagen se sube a *Firebase Storage*. La función descarga la imagen y crea una versión en miniatura. La función escribe la ubicación de esa miniatura en la base de datos, de manera que una aplicación cliente pueda encontrarla y usarla. La función vuelve a subir la miniatura a *Firebase Storage* en una ubicación nueva. La aplicación descarga el vínculo de la miniatura.

Realizar integraciones con API y servicios de terceros.

Cloud Functions puede ayudar a que una aplicación funcione con servicios de terceros a través de llamadas API web. Por ejemplo, una aplicación que se usa para la programación colaborativa podría notificar las confirmaciones de GitHub a un grupo de trabajo.



Un usuario envía confirmaciones a un repositorio de GitHub. Una función HTTPS se activa a través de la API de *webhook* de GitHub. La función envía una notificación de la confirmación al equipo de trabajo.

5.1.2. Configurar Cloud Functions

Firebase Cloud Functions utiliza *Firebase CLI* para programar funciones. Se realiza de forma parecida a como lo utilizamos con *Firebase Hosting*. También hay que inicializar *Cloud Functions* en el proyecto de *Firebase*.

En primer lugar, instala *Firebase CLI*, si has realizado el apartado de *Firebase Hosting* de la unidad anterior, ya debes tenerlo instalado.

Ejercicio: *Inicializar un proyecto Cloud Functions en Firebase CLI.*

1. Instala *Firebase CLI*:

   Firebase CLI (*Command Line Interface*) requiere *Node.js* y *npm*. Puedes instalar ambos siguiendo las instrucciones en *https://nodejs.org*. Instalando *Node.js* también se instala *npm*. *Firebase CLI* necesita la versión de *Node.js* 0.10.0 o superior.

   Una vez instalado *Node.js* y *npm*, puedes instalar *Firebase CLI* vía *npm*, ejecuta desde la línea de comandos:

```
npm install -g firebase-tools
```

   Para actualizar a la última versión, simplemente ejecuta el mismo comando.

2. Loguéate en *Firebase*:

   Ejecuta el comando `firebase login`. Si tienes una sesión de *Google* activa y que se pueda conectar a *Firebase Console*, debe devolver el mensaje: *"Already logged in as xxxx@gmail.com"*. Si no es así, se lanzará el navegador predeterminado para que inicies la sesión en una cuenta de *Google* y autorices el acceso a *Firebase* desde *Firebase CLI*. Puedes cerrar la sesión ejecutando `firebase logout`.

3. Inicializa la aplicación:

   Abre la consola del sistema operativo y sitúate en el directorio del ordenador donde tienes o vas a desarrollar el proyecto de *Firebase*, ejecutando el comando `cd`. En nuestro caso accede a la carpeta *"Eventos"* creada anteriormente, por ejemplo, con `cd c:\eventos` en *Windows*. Posteriormente, ejecuta el comando `firebase init`:

```
cd ruta_carpeta_proyecto
firebase init
```

   El comando `firebase init` lanza un asistente de configuración del proyecto. Hemos de tener en cuenta que ya hemos inicializado el proyecto para *Firebase Hosting*. Vamos a ver como añadir una nueva función a un proyecto ya inicializado:

```
You're about to initialize a Firebase project in this directory:
  C:\eventos
Before we get started, keep in mind:

  * You are initializing in an existing Firebase project directory

? Are you ready to proceed? (Y/n)
```

   El asistente nos advierte de que ya existe un proyecto *Firebase* en el directorio. Introduce `y` y pulsa la tecla `INTRO` para indicar que estás de acuerdo en proceder con la configuración del proyecto.

```
? Which Firebase CLI features do you want to setup for this folder? Press
Space to select features, then Enter to confirm your choices.
 ( ) Database: Deploy Firebase Realtime Database Rules
 ( ) Firestore: Deploy rules and create indexes for Firestore
>(*) Functions: Configure and deploy Cloud Functions
 (*) Hosting: Configure and deploy Firebase Hosting sites
 ( ) Storage: Deploy Cloud Storage security rules
```

Nos preguntan qué características de *Firebase* queremos activar. Tenemos que elegir las que ya tenemos inicializadas (*Hosting*) y la nueva (*Functions*). Marca las dos opciones. Pulsa la tecla `INTRO` para continuar.

```
=== Project Setup

First, let's associate this project directory with a Firebase project.
You can create multiple project aliases by running firebase use --add,
but for now we'll just set up a default project.

i  .firebaserc already has a default project, skipping
```

Como ya tenemos un proyecto de *Firebase* asociado al proyecto en *Firebase CLI*, no nos pregunta que proyecto queremos asociar. Ya lo hemos hecho en el apartado de *Firebase Hosting*.

```
=== Functions Setup

A functions directory will be created in your project with a Node.js
package pre-configured. Functions can be deployed with firebase deploy.

? What language would you like to use to write Cloud Functions?
>  JavaScript
  TypeScript
```

A partir de este apartado pasaremos a configurar *Firebase Functions*, seguidamente nos pregunta por la configuración específica de *Firebase Functions*. Primero nos pregunta por el lenguaje de programación que vamos a utilizer, podremos elegir entre *JavaScript* y *TypeScript*. Vamos a elegir *JavaScript*.

```
? Do you want to use ESLint to catch probable bugs and enforce style?
(y/N)
```

Elegimos que no queremos utilizer ESLint.

```
+  Wrote functions/package.json
+  Wrote functions/index.js
? Do you want to install dependencies with npm now? (Y/n)
```

El asistente creará un directorio nuevo llamado *functions* y dos ficheros: *package.json* y *index.js*. Nos pregunta si queremos instalar las dependencias, le indicamos que sí, introduciendo `Y` y pulsando `INTRO`. Esperamos un momento mientras se instalen las dependencias.

A partir de este punto, tenemos que volver a configurar *Firebase Hosting* porque forma parte de nuestro proyecto.

```
=== Hosting Setup

Your public directory is the folder (relative to your project directory)
that will contain Hosting assets to be uploaded with firebase deploy. If
you have a build process for your assets, use your build's output
directory.

? What do you want to use as your public directory? (public)
```

Indicamos cual será la carpeta que contendrá los archivos que se subirán a *Firebase Hosting*. Por defecto, su nombre será *"public"*. Dejaremos este nombre y seguiremos adelante, pulsando la tecla `INTRO`. Es lo que hemos indicado en la anterior instalación y es conveniente mantener la misma configuración:

```
? Configure as a single-page app (rewrite all urls to /index.html)? (y/N)
```

Tal y como hemos hecho en la anterior inicialización, elegiremos que no queremos configurar una sola página para la aplicación. Pulsa la tecla `INTRO` para continuar.

```
? File public/404.html already exists. Overwrite? (y/N)
```

Indica que no quieres sobrescribir la página *404.html*.

```
? File public/index.html already exists. Overwrite? (y/N)
```

Tampoco queremos sobrescribir el fichero *index.html*. Ya lo tenemos implementado desde el apartado de Hosting.

```
+  Firebase initialization complete!
```

Hemos terminado la inicialización del proyecto en *Firebase CLI*. La ejecución del comando `firebase init` crea varios archivos y carpetas en el directorio del proyecto. Sirven para configurar e implementar el proyecto.

Después de inicializar el proyecto para *Functions*, la estructura del proyecto será similar a la siguiente:

```
proyecto
 +- .firebaserc # Archivo oculto que ayuda a cambiar de forma rápida
 |              # entre proyectos con: firebase use
 |
 +- firebase.json # Describe propiedades del proyecto
 |
 +- functions/ # Directorio que contiene el código de las funciones.
      |
      +- package.json  # Fichero del paquete npm que
      |                # describe el código de Cloud Functions
      +- index.js   # Fichero principal de código de Cloud Functions
      |
      +- node_modules/ # Directorio donde estan instaladas
                       # las dependencias (declaradas en package.json)
```

Nuestra carpeta de proyecto contendrá más directorios y archivos, ya que hemos configurado una nueva caraterística de *Firebase* en el proyecto. Ya podemos comenzar a agregar código. Lo haremos en el fichero *index.js* que encontrarás en la carpeta *functions*. El lenguaje de programación que debemos utilizar es *JavaScript*.

Una vez implementadas las funciones en el fichero *index.js*, deberemos subir la función a los servidores de *Firebase*. Para ello utilizaremos el comando `firebase deploy --only functions`. `firebase deploy` ya lo hemos utilizado en *Firebase Hosting* para sincronizar el proyecto local con *Firebase*. Añadimos `-only functions` si queremos que sólo sincronice las funciones.

Para programar las funciones, debemos importar los módulos necesarios. Vamos a ver como escribir el código de las funciones.

5.1.3. Escribir funciones

Las funciones las escribiremos en el fichero *index.js* del proyecto.

Debemos importar los módulos que se necesiten mediante el método `require`. Será obligatorio incluir `require` para el SDK de *Firebase Cloud Functions*. Nos permite crear funciones *Cloud Functions* y configurar los desencadenadores. Lo haremos incluyendo la siguiente línea en el fichero *index.js*:

```
const functions = require('firebase-functions');
```

Además, dependiendo de la funcionalidad de las funciones a programar, deberemos importar diferentes módulos. Por ejemplo, `require('firebase-admin')` será necesario si queremos ejecutar acciones como: leer o escribir datos en *Realtime Database*, enviar mensajes de *Firebase Cloud Messaging*, generar y verificar *tokens* de autentificación de *Firebase* o para crear una consola de administrador simplificada.

Firebase CLI instala de forma automática los módulos de *Node* de *Firebase* y del SDK de *Firebase* para *Cloud Functions* cuando se inicializa un proyecto. Para agregar bibliotecas de terceros al proyecto, puedes modificar el fichero *functions/package.json*, ejecutar `npm install` y realizar solicitudes.

Nos quedaría agregar el código propio de las funciones. Distinguimos tres partes en cada función: nombre de la función, desencadenador asociado y código a ejecutar.

```
exports.nombreFuncion = functions.desencadenador((req, res) => {
   //Código a ejecutar
});
```

Definiremos el nombre de las funciones con la instrucción `exports.` seguido del nombre que queramos.

Tienes que asociar un objeto a cada desencadenador. También tendrás que indicar que evento del objeto va a ser el que va a activar la función. Por ejemplo, asociamos a una función un objeto *Realtime Database* e indicamos que se va a activar cuando se produzca un evento de escritura. En la siguiente tabla puedes ver

los desencadenadores y los objetos asociados, posteriormente veremos los eventos de cada objeto:

Desencadenadores	Objeto
Cloud Firestore	`functions.firestore`
Realtime Database	`functions.database`
Authentication	`functions.auth`
Analytics	`functions.analytics`
Crashlytics	`functions.crashlytics`
Cloud Storage	`functions.storage`
HTTP	`functions.https`
Cloud Pub/Sub	`functions.pubsub`

5.1.3.1. Activadores de Cloud Firestore

Los eventos que se disparan cuando realizamos cambios en los datos de *Cloud Firestore*, son los que ejecutan las funciones de Firebase Functions.

El ciclo de vida de una función *Cloud Firestore* es el siguiente:

1. Espera a que ocurran cambios en un documento en particular.
2. Se activa cuando ocurre un evento implementado en la función.
3. Recibe un objeto de datos del evento que contiene dos instantáneas de los datos almacenados: una con los datos originales previos al cambio y una con los datos nuevos.

Un objeto `functions.firestore` para *Cloud Firestore* admite los siguientes eventos:

Evento	Descripción
`onCreate`	Se activa cuando se escribe en un documento por primera vez.
`onUpdate`	Se activa cuando un documento ya existe y se cambia uno de sus valores.
`onDelete`	Se activa cuando se borra un documento con datos.
`onWrite`	Se activa cuando se activa `onCreate`, `onUpdate` u `onDelete`.

Los eventos de *Cloud Firestore* se activan solo con los cambios en documentos. No es posible agregar eventos a campos específicos.

Por ejemplo, para activar un evento para cualquier cambio en un documento específico, puedes usar la función `functions.firestore.document('…').onWrite(…)`.

```
// Escuchador para cualquier cambio en el documento 'fichero'
// de la colección 'empresa'
exports.nombreFuncion = functions.firestore
                .document('empresa/fichero').onWrite((data, context) => {
```

```
    // ... Aquí el código
  });
```

Cuando se activa una función, es posible obtener los datos actualizados del documento y los previos a la actualización. También podemos obtener el contexto en el cual se ha disparado del evento.

Para obtener los datos afectados por la función consultaremos el parámetro `data`. Los datos actuales del documento los obtendremos mediante `data.after.data()` y los datos anteriores a la actualización con `data.before.data()`.

```
exports.actualizarUsuario = functions.firestore.document('users/{userId}')
                                        .onUpdate((data,context) => {
    // Obtenemos el objeto que contiene el documento actual
    var newValue = data.after.data();
    // Obtenemos el objeto que contiene el documento anterior
    var previousValue = data.before.data();
   });
```

Con un objeto `data` podemos acceder a las propiedades del mismo modo que con cualquier otro objeto. También puedes utilizar la función `get` para acceder a campos específicos, por ejemplo: `data.after.data().edad` o `data.after.data()['nombre']`.

Cada invocación de función está asociada a un documento específico en la base de datos de *Cloud Firestore*. Puedes acceder a ese documento con `DocumentReference` llamando a `data.after.ref`. `DocumentReference` incluye métodos como `update()`, `set()` y `remove()` para que puedas modificar con facilidad el documento que activó la función. Cada vez que se escribe en el mismo documento que activó una función, se corre el riesgo de crear un bucle infinito. Hay que tener cuidado y asegurase de salir de forma segura de la función cuando no necesites hacer cambios.

Si no se conoce el `ID` del documento específico al que deseas adjuntar un activador, se puede usar un `{wildcard}`.

```
exports.useWildcard = functions.firestore.document('eventos/{evento}')
                                       .onWrite((data, context) => {
    // Código a ejecutar
  });
```

En este ejemplo, cuando se cambia algún campo en cualquier documento dentro de `eventos`, la función será activada y en `evento` se nos indicará el nombre del documento. Si un documento de `eventos` tiene subcolecciones y se cambia un campo en uno de los documentos de esas subcolecciones, la función no se activa.

Las coincidencias de comodines se extraen de la ruta del documento y se almacenan en `context.resource` en el campo `name`. Puedes definir tantos comodines como desees para sustituir los `ID` explícitos de colección o de documento. En el siguiente código puedes ver un ejemplo de una función con

múltiples comodines.

```
exports.useWildcards = functions.firestore
            .document('eventos/{paises}/{ciudades}/{evento}/')
            .onWrite((data, context) => {
    //Código a ejecutar
  });
```

El activador siempre debe apuntar a un documento, incluso si usas un comodín. Por ejemplo, `eventos/{paises}/{ciudades}` no es válido porque `{ciudades}` es una colección. Sin embargo, `eventos/{paises}/{ciudades}/{evento}` es válido porque `{evento}` siempre apuntará a un documento.

Puedes consultar el contexto en el que se ha disparado una función. Para ello consulta el parámetro `context`. Con `context.resource` obtendrás la fuente que ha disparado la función: el servicio y el nombre del objeto que ha provocado el disparo. Un ejemplo de los valores de `context.resource`:

```
{service: 'firestore.googleapis.com',
 name:'projects/eventos/databases/(default)/documents/eventos/carnaval'
}
```

Usa `context.eventType` para consultar el tipo de evento que ha disparado la función. En el siguiente ejemplo, puedes ver que ha sido la escritura en un documento de *Firestore* el que lo ha disparado:

```
google.firestore.document.write
```

Vamos a ver un ejemplo en la aplicación “Eventos”. Al modificar una imagen de un evento, se lanzará una notificación *Firebase Cloud Messaging* a los usuarios de la aplicación anunciando el cambio. Utilizaremos *Firebase Functions* para *Firestore*, ya que, al subir una nueva imagen para un evento, se actualiza la ruta a la nueva imagen en la base de datos y eso será el evento activador.

Ejercicio: *Activador Firestore en Firebase Functions.*

1. Abre el fichero *index.js* del directorio *functions* del proyecto *Eventos* que has creado con Firebase CLI con un editor de texto y elimina todas las líneas comentadas (las que empiezan por //).
   El fichero contendrá solamente la línea: `const functions = require('firebase-functions');`. De esta forma importamos el SDK de *Firebase* para *Cloud Functions* para crear funciones y configurar los desencadenadores.
2. Añade el siguiente código al final del fichero:

```
const admin = require('firebase-admin');
admin.initializeApp(functions.config().firebase);
```

Importamos el SDK de *Firebase Admin* para acceder a *Firestore* y *Firebase Cloud Messaging*, y lo inicializamos.

3. Inserta la siguiente función al final del fichero:

```
var tema = "Todos";
exports.notificaFoto = functions.firestore.document('eventos/{evento}')
                                          .onWrite((data, context) => {
   if         (data.after.data()['imagen']!=data.before.data()['imagen']){
      const datos = {
         notification: {
            title: 'Nueva imagen de '+ data.after.data()['evento'],
               body: "Se ha cambiado la imagen del evento "
               + data.after.data()['evento']
         }
      };
      admin.messaging().sendToTopic(tema, datos)
         .then(function(response) {
            console.log("Envío correcto:", response);
         })
         .catch(function(error) {
            console.log("Envío erróneo:", error);
         });
   }
});
```

4. Accede a la consola de comandos y posicionado en la carpeta del proyecto, ejecuta la siguiente instrucción para implementar la función en *Firebase*:

```
firebase deploy --only functions
```

5. Accede a la consola de Firebase. En el apartado *"Develop"* selecciona *"Functions"*. Comprueba que la función está dada de alta.
6. Cambia la imagen de un evento desde la aplicación y comprueba que recibes una notificación.

Comprueba que has recibido una notificación al realizar el cambio en la imagen de un evento. Al tener la aplicación abierta verás un cuadro de diálogo, no una notificación en el área de notificaciones del dispositivo.

Hemos creado una función de Firebase llamada `notificaFoto`. Se activará cuando cuando se haga un cambio en el campo `imagen` de un documento de Firestore, para ello indicamos que `functions.firestore.document` será el objeto que desencadenará la función. Con `'eventos/{evento}'` expresamos que cualquier cambio en un documento de la colección `eventos` activará la función. Con `onWrite` implementamos que será cualquier cambio el que desencadenará la ejecución del código de la función. Sólo queremos que se lance la notificación cuando se cambie el campo `imagen` de cualquier documento, si se cambia cualquier otro campo no queremos que avise. La comparación del valor actual del campo (`data.after.data()['imagen']`) con el anterior (`data.before.data()['imagen']`) determinará si se lanza o no la notificación.

Enviamos las notificaciones a un tema determinado mediante el método `admin.messaging().sendToTopic`. Necesitamos especificar el tema al cual enviar

la notificación (vamos a enviarlo al tema `'Todos'`) y los datos de la notificación. Los datos de la notificación los especificamos mediante un objeto `Notification` rellenando los campos `title` y `body`.

Otra forma de probar que funciona la función en *Firestore* es cambiar el valor del campo *"imagen"* en algún evento. Realiza un cambio y comprueba que al guardar los cambios recibes una notificación. Vuelve a poner el valor original o no se verá la imagen en la aplicación.

5.1.3.2. Activadores de Realtime Database

Permite ejecutar operaciones en los servidores de Google que se activan cuando ocurren ciertos eventos en Firebase Realtime Database.

Para crear funciones `functions.database` para eventos *Realtime Database*, debes especificar que evento activa la función y la ruta de acceso a la base de datos.

Las funciones permiten controlar los eventos específicos de creación, actualización o eliminación, o detectar cambios de cualquier tipo. Los eventos disponibles son:

Evento	Descripción
`onCreate`	Se activa cuando se crean nuevos datos.
`onUpdate`	Se activa cuando se actualizan datos.
`onDelete`	Se activa cuando se borran datos.
`onWrite`	Se activa cuando se activa `onCreate`, `onUpdate` u `onDelete`.

Para controlar cuándo se debe activar la función, implementa `ref(path)`. Este método especifica la ruta de acceso dentro de la base de datos que activará la función. Por ejemplo:

```
functions.database.ref('/foo/bar')
```

Los eventos de una ruta afectan a la ruta, incluidas aquellas que ocurren en cualquier lugar dentro de ella. Por ejemplo, si configuras la ruta de acceso `/foo/bar` para la función, se muestran coincidencias de eventos en estas dos ubicaciones:

```
/foo/bar
/foo/bar/baz/really/deep/path
```

En ambos casos, *Firebase* interpreta que el evento ocurre en `/foo/bar`. Los datos del evento incluyen los datos antiguos y actualizados en `/foo/bar`. Si la cantidad de datos que disparan el evento son grandes, es mejor usar varias funciones con rutas de acceso más profundas en lugar de una sola función cerca de la raíz. Para mejorar el rendimiento, solicita que los datos que disparen la función estén en el nivel más profundo posible.

Puedes especificar una ruta de acceso con comodines. Encierra entre corchetes el comodín, por ejemplo: `ref('foo/{bar}')`, coincide con cualquier elemento secundario de `/foo`. Los valores comodín están disponibles en la función con del

objeto `context.resource`.

Las rutas de acceso con comodines pueden coincidir con varios eventos de una misma escritura. Por ejemplo, la inserción de:

```
{
  "foo": {
    "hola": "mundo",
    "firebase": "functions"
  }
}
```

Coincide con la ruta de acceso `"/foo/{bar}"` dos veces: una vez con `"hola": "mundo"` y otra con `"firebase": "functions"`.

Al igual que con *Firestore*, podemos recuperar los datos originales y los modificados mediante un objeto `data`. El objeto recuperado es un `DeltaSnapshot`. Para leer los datos antes de la modificación usa `data.before` y para leer los datos tras la modificación usa `data.after`.

En algunos casos, no necesitas el valor antiguo y solo necesitas saber si los datos se cambiaron. Para esto usa el método `changed()` del objeto `DeltaSnapshot`.

5.1.3.3. Activadores de Firebase Authentication

Puedes activar una función en respuesta a la creación o la eliminación de cuentas de usuario de *Firebase Authentication*.

Los eventos disponibles para el objeto `functions.auth.user` son:

Evento	Descripción
`onCreate`	Se crea un usuario.
`onDelete`	Se elimina un usuario.

Por ejemplo, cuando se crea un usuario podríamos enviar un correo electrónico de bienvenida. Para ello usarìamos el evento `onCreate()`:

```
exports.enviarEmailBienvenida = functions.auth.user().onCreate((user) => {
  // Código a ejecutar
});
```

Firebase activará eventos de creación de usuarios en los siguientes casos:

- Un usuario crea una cuenta de correo electrónico y una contraseña.
- Un usuario accede por primera vez con un proveedor de identidad federada.
- El programador crea una cuenta con el SDK de administrador de Firebase.
- Un usuario accede a una sesión de autenticación anónima por primera vez.

Cloud Functions no activa ningún evento cuando un usuario accede por primera vez con un token personalizado.

A través del parámetro `user`, puedes acceder a los atributos de un usuario

recién creado con el objeto `UserRecord`. Por ejemplo, con `user.email` obtenemos el correo electrónico y con `user.displayName` conseguiremos el nombre visible del usuario.

Cuando se elimina un usuario, se activan los eventos de eliminación de usuarios. Debes usar el controlador de eventos `functions.auth.user().onDelete()`:

```
exports.enviarEmail = functions.auth.user().onDelete((user) => {
  // Código a ejecutar
});
```

5.1.3.4. Activadores de Google Analytics para Firebase

Google Analytics para *Firebase* proporciona informes de eventos que te ayudan a comprender la forma en que los usuarios interactúan con una aplicación. Con *Cloud Functions*, puedes crear funciones que se activen con eventos de conversión.

Los eventos disponibles son `functions.analytics.event`:

Evento	Descripción
`onLog`	Se produce una conversión.

Actualmente, *Cloud Functions* solo admite eventos marcados como eventos de conversión. Puedes especificar cuáles son los eventos de conversión en la pestaña "*Eventos"* del panel *Analytics* de la consola de *Firebase*.

Cloud Functions admite ejecutar funciones con la activación de eventos `AnalyticsEvent` de *Google Analytics*. Se activan cada vez que la actividad del usuario genera un evento de conversión. Por ejemplo, podríamos recibir un email cuando se genere el evento `in_app_purchase`, para indicar que se ha producido una compra desde la aplicación. Debes especificar el evento de *Analytics* que deseas que active la función con el método `functions.analytics.event()` y supervisar el evento dentro del controlador de evento `onLog()`:

```
exports.enviarEmailCompra =
          functions.analytics.event('in_app_purchase').onLog((evento) => {
      // Código a ejecutar
   });
```

Con cada evento de *Analytics*, tienes acceso a todos los parámetros y las propiedades de relevantes del usuario. Incluye información sobre el usuario, el dispositivo, la aplicación y la información geográfica del evento. Para obtener una lista completa de parámetros y propiedades del usuario, consulta la referencia `functions.analytics`.

Observa este ejemplo, de una función activada por una compra, que recoge diversos parámetros del usuario, aplicación y dispositivo:

```
exports.enviarCuponPorCompra =
         functions.analytics.event('in_app_purchase').onLog((evento) => {
      const user = event.user;
```

```
      // Id de usuario.
      const uid = user.userId;
      // Cantidad compra en $.
      const purchaseValue = event.valueInUSD;
      // Lenguaje del usuario
      const userLanguage = user.deviceInfo.userDefaultLanguage;
      // Enviar cupon via notificación FCM
      return enviarCuponViaFCM(uid, userLanguage);
});
```

Seguidamente vamos a implementar en la aplicación “Eventos” una función que nos avise por email cuando un usuario desinstale la aplicación.

Ejercicio: *Activador Google Analytics en Firebase Functions.*

1. Accede a la consola de Firebase y selecciona el proyecto “Eventos”.
2. En el apartado “Analytics” selecciona la opción “Events”.
3. Activa como un “Evento de conversión” el evento `app_remove`.
   Recuerda que solamente podemos asociar funciones a eventos de conversión.
4. Abre el fichero *index.js* del directorio *functions* del proyecto *Eventos* que has creado con Firebase CLI con un editor de texto y añade el siguiente código:

```
const nodemailer = require('nodemailer');
exports.enviarEmailDesinstalacion =
              functions.analytics.event('app_remove').onLog((event) => {
   const gmailEmail = functions.config().gmail.email;
   const gmailPassword = functions.config().gmail.password;
   const mailTransport = nodemailer.createTransport({
      service: 'gmail',
      auth: {
         user: gmailEmail,
         pass: gmailPassword
      }
   });
   const opcionesEmail = {
      from: '${APP_NAME} <noreply@firebase.com>',
      to: 'tucorreo@correo.com',
      subject: 'Desinstalación aplicación Eventos',
      text: 'Un usuario ha desinstalado la aplicación Eventos'
   };
   return mailTransport.sendMail(opcionesEmail);
});
```

Sustituye `tucorreo@correo.com` por la dirección del email que va a recibir los correos.

Hemos creado una función llamada `enviarEmailDesinstalacion` y asociado a un evento de desinstalación de la aplicación mediante `functions.analytics.event('app_remove')`. Enviamos un correo electrónicio desde la función, para ello utilizamos `nodemailer`. Enviamos los emails mediante una cuenta de *Gmail*, pero tenemos que tener en cuenta las limitaciones de envío de *Gmail*. Si tienes unas necesidades más grandes considera utilizar cuentas profesionales como *Sendgrid*, *MailChimp*,…

5. Para habilitar el envío de correo desde una cuenta de *Gmail* tienes que permitir el acceso a la cuenta desde aplicaciones menos seguras y desbloquear el *Captcha* de la cuenta. Para ello accede a las siguientes direcciones y realiza las acciones pertinentes:

   Habilita el acceso de aplicaciones menos seguras:
   https://www.google.com/settings/security/lesssecureapps

   Desbloquea el captcha de la cuenta:
   https://accounts.google.com/DisplayUnlockCaptcha

6. Abre una ventana con la consola de comandos y sitúate en el directorio *functions* de la carpeta del proyecto *"Eventos"* que has implementado en *Firebase CLI*.
7. Ejecuta la siguiente instrucción en la consola de comandos para actualizar todas las dependencias del proyecto:

```
npm install
```

   Asegúrate de estar situado en la carpeta *functions* cuando ejecutes la instrucción.

8. Configura el valor de las variables Google Cloud (gmail.email y gmail.password) con el email y la contraseña de la cuenta de Gmail que será usada para enviar los correos electrónicos. Ejecuta la siguiente instrucción en la consola de comandos:

```
firebase functions:config:set gmail.email="cuenta@gmail.com"
gmail.password="contraseña"
```

   Sustituye `"cuenta@gmail.com"` por la cuenta de Gmail que enviará los correos y `"contraseña"` por la contraseña de la cuenta.

9. Despliega el proyecto en la consola de *Firebase* con la siguiente instrucción:

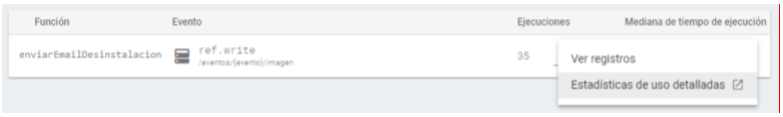
```
firebase deploy --only functions
```

   Si devuelve el siguiente error; `"Error: Error parsing triggers: Cannot find module 'nodemailer'"`, ejecuta la siguiente instrucción `npm install nodemailer --save` y añade `"nodemailer": "^6.4.2"` a las dependencias en el fichero *package.json* de la carpeta *functions*.

10. Prueba el desencadenador.

    Puedes probar el desencadenador de dos formas: haciendo una prueba desde la consola de Firebase o desinstalando la aplicación en un dispositivo.

    Podemos activar la función desde la consola de Firebase para realizar pruebas. Desde la consola de Firebase, selecciona "Functions". En la función "enviarEmailDesinstalacion", selecciona la opción "Estadísticas de uso detalladas".



Selecciona la pestaña “Prueba” en la nueva ventana que se ha abierto. Pulsa en el botón “Prueba la función”.

Esta técnica te puede servir para probar de forma rápida algunas funciones y sus efectos en la aplicación.

Para probar desde el dispositivo simplemente desinstala la aplicación “Eventos” desde los ajustes del dispositivo.

En ambos casos vas a recibir un correo electrónico avisando de que un usuario ha desinstalado la aplicación.

5.1.3.5. Activadores de Firebase Crashlytics

Puedes implementar funciones en respuesta a eventos *Crashlytics*. Estos eventos pueden ser por nuevos problemas, problemas regresivos o alertas de velocidad. Por ejemplo, podríamos notificar a nuestro equipo de desarrollo cuando se produzca un problema de velocidad.

Los eventos disponibles son:

Evento	Descripción
`onNew`	Se produce un problema por primera vez.
`onRegressed`	Se experimenta un problema que ya se ha producido.
`onVelocityAlert`	Un dispositivo presenta problemas de velocidad.

Para crear una función *Crashlytics*, primero generaremos un objeto `IssueBuilder` con `functions.crashlytics.issue()`, entonces llamaremos a la función apropiada.

En el siguiente ejemplo, vemos como la función se activa cuando la aplicación experimenta un problema por primera vez:

```
exports.nuevoError=functions.crashlytics.issue()
                                        .onNew(async (issue) => {
   // Código a ejecutar
});
```

También podemos capturar eventos que ya se han producido con `onRegressed`

```
exports.errorConocido = functions.crashlytics.issue()
                                        .onRegressed(async (issue) => {
   // Código a ejecutar
});
```

Por último, indicar que también podemos capturar eventos de velocidad. Se producen cuando el dispositivo presenta problemas de velocidad:

```
exports.alertaVelocidad = functions.crashlytics.issue()
                                    .onVelocityAlert(async (issue) => {
   // Código a ejecutar
});
```

Cada evento activado creado con `IssueBuilder` retorna un problema. Un problema tiene propiedades, que incluyen: nombre del problema, un identificador, información relevante y mucho más. Por ejemplo, para obtener el identificador de un problema utilizaríamos `issue.issueId`, y para el título `issue.issueTitle`.

5.1.3.6. Activadores de Cloud Storage

Activa una función en respuesta a la carga, actualización o eliminación de archivos y carpetas en *Cloud Storage*. Puedes definir el alcance de la función: para un depósito específico de *Cloud Storage* o utilizar el depósito predeterminado.

Los eventos disponibles son:

Evento	Descripción
`onArchive`	Solo se envía cuando un depósito habilita el control de versiones. La versión publicada se convirtió en una versión archivada, bien porque se la archivó o porque se reemplazó cuando se subió un objeto con el mismo nombre.
`OnDelete`	Se envía cuando un objeto se borra permanentemente.
`onFinalize`	Se envía cuando un nuevo objeto se crea correctamente.
`onMetadataUpdate`	Se envía cuando cambian los metadatos de un objeto existente.

Veamos un ejemplo del alcance de una función:

- `functions.storage.object()` detecta cambios de objetos en el depósito de almacenamiento predeterminado.
- `functions.storage.bucket('depositoA').object()` para detectar los cambios de objetos en un depósito específico.

El siguiente código muestra una función que se activa si se produce un evento en el depósito de almacenamiento predeterminado:

```
exports.nombreFuncion = functions.storage.object()
                                    .onFinalize(async (object) => {
   // Código a ejecutar
});
```

En el ejemplo anterior hemos utilizado el evento `onFinalize`. Se activa cuando crea un objeto dentro de un depósito.

Cloud Functions obtiene varios atributos del objeto modificado en `object`, como, por ejemplo: `timestamp`, `resource`, `size` y `contentType`. El atributo `resourceState` tiene el valor `"exists"` (si se ha creado o modificado un objeto) o `"not_exists"` (si se ha eliminado o movido). El atributo `resourceState` debe sincronizarse con el atributo `metageneration` para saber si un objeto acaba de crearse. Es un contador que aumenta cuando se produce un cambio en los metadatos del objeto. En un objeto nuevo, su valor es 1.

Veamos un ejercicio en el que vamos a generar imágenes en miniatura a partir de las imágenes subidas desde la aplicación *"Eventos"*.

Ejercicio: *Crear miniatura al subir imagen a Cloud Storage.*

1. Abre el fichero *index.js* del directorio *functions* del proyecto "*Eventos"* y añade la siguiente función al final del fichero:

```
exports.crearMiniatura =
          functions.storage.object().onFinalize(async (object) => {
   // Código a ejecutar
});
```

2. Declara las siguientes variables debajo de la línea `// Codigo a ejecutar` :

```
const deposito = object.bucket;
const ruta = object.name;
const tipo = object.contentType;
const estado = object.resourceState;
const metageneration = object.metageneration;
```

   El parámetro `object` contiene el objeto *Storage* que ha activado la función. El deposito que contiene el fichero lo guardamos en la variable `deposito` [`object.bucket`]. Guardamos en la variable `ruta` [`object.name`] donde se encuentra el fichero. `tipo` [`object.contentType`] contiene el tipo de fichero modificado. `estado` [`object.resourceState`] nos informa de si es un alta o modificación (`'exists'`) o bien de un borrado o que se ha movido el fichero (`'not_exists'`). En la variable `metageneration` [`object.metageneration`] guardamos el número de veces que han cambiado los metadatos del archivo, siendo 1 el valor para objetos nuevos.

3. Añade debajo del código anterior el siguiente:

```
// Salimos del desencadenador si el fichero no es una imagen.
if (!tipo.startsWith('image/')) {
```

```
  console.log('No es una imagen.');
  return null;
}
// Obtenemos el nombre del fichero.
const nombreFichero = path.basename(ruta);
// Salimos de la función si es una miniatura.
if (nombreFichero.startsWith('thumb_')) {
  console.log('Existe una miniatura.');
  return null;
}
// Salimos de la función si se trata de un evento mover o borrado.
if (estado === 'not_exists') {
  console.log('Evento de borrado.');
  return null;
}
// Salimos si no es un fichero nuevo. Sólo ha cambiado los metadata.
if (estado === 'exists' && metageneration > 1) {
  console.log('Cambio de metadata.');
  return null;
}
```

Hemos definido todos los casos en los que no se va a generar la miniatura de una imagen: el fichero no sea una imagen, si es una miniatura, si es un borrado,… Lo que haremos será transformar una imagen en miniatura si se trata de un fichero imagen nuevo en *Cloud Storage*.

4. Para descargar y volver a subir objetos con facilidad en *Cloud Storage*, desde la instancia del evento, instala el paquete de *Google Cloud Storage* ejecutando la siguiente instrucción en la consola de comandos, situado en la carpeta *functions*:

```
npm install --save @google-cloud/storage
```

5. Inserta al inicio del archivo *index.js* las siguientes importaciones:

```
const gcs = require('@google-cloud/storage');
const spawn = require('child-process-promise').spawn;
const path = require('path');
const os = require('os');
const fs = require('fs');
```

En un ejercicio anterior ya hemos importado el requerimiento más importante de las funciones (`'firebase-functions'`). Para usar promesas de *JavaScript* importamos `'child-process-promise'`. También importaremos `'@google-cloud/storage'`, `'path'`, `'os'` y `'fs'`, para descargar y realizar el procesamiento de ficheros.

6. Transforma una imagen subida en una miniatura en la función, para ello debes insertar el siguiente código al final de la función `crearMiniatura`:

```
const bucket = admin.storage().bucket(deposito);
const tempRuta = path.join(os.tmpdir(), nombreFichero);
const metadata = {
```

```
    contentType: tipo,
};
return bucket.file(ruta).download({
    destination: tempRuta,
}).then(() => {
    return spawn('convert', [tempRuta, '-thumbnail', '200x200>', tempRuta]);
}).then(() => {
    const thumbArchivo = `thumb_${nombreFichero}`;
    const thumbRuta = path.join(path.dirname(ruta), thumbArchivo);
    return bucket.upload(tempRuta, {
        destination: thumbRuta,
        metadata: metadata,
    });
}).then(() => fs.unlinkSync(tempRuta));
```

7. Instala el paquete de *child-process-promise* ejecutando la siguiente instrucción en la consola de comandos, situate en la carpeta *functions*:

```
npm install child-process-promise --save
```

8. Ejecuta desde la consola y posicionado en la carpeta del proyecto la siguiente instrucción para implementar la función en *Firebase*:

```
firebase deploy --only functions
```

9. Accede a la consola de *Firebase*. En el apartado *"Develop"* selecciona *"Functions"*. Comprueba que la función está dada de alta.
10. Cambia imagen de un evento desde la aplicación.

Comprueba que recibes una notificación en la aplicación y que la imagen subida ha reducido su tamaño.

Hay casos, en que no sea necesario descargar archivos desde *Cloud Storage*. Sin embargo, para realizar tareas intensivas, como generar una miniatura, debes descargar el archivo en la instancia de la función (en la máquina virtual que ejecuta el código).

Utiliza `gcs.bucket.file(filePath).download` para descargar un archivo en un directorio temporal en la instancia de *Cloud Functions*. En esta ubicación, puedes procesar el archivo según sea necesario y luego subirlo a *Cloud Storage*. Cuando ejecutes tareas asíncronas, asegúrate de mostrar una promesa de *JavaScript*.

Cloud Functions proporciona un programa de procesamiento de imágenes llamado `ImageMagick`. Puede manipular archivos de imágenes.

El código ejecuta el método `convert` del programa `ImageMagick` para crear una miniatura de 200 x 200 píxeles de la imagen original en un directorio temporal y después la sube a *Cloud Storage*.

5.1.3.7. Activadores HTTP

Puedes activar una función a través de una solicitud de HTTP mediante `functions.https`. Permite invocar una función síncrona a través de los siguientes métodos HTTP admitidos: `GET`, `POST`, `PUT`, `DELETE` y `OPTIONS`.

El evento disponible es:

Evento	Descripción
`onRequest`	Llega una solicitud HTTP.

Vamos a ver un ejemplo de una función `GET` de HTTP. La función recupera la hora actual del servidor, formatea el tiempo según se especifica en un parámetro de consulta URL y envía el resultado en la respuesta HTTP.

Usa `functions.https` para crear una función que controle eventos HTTP. El controlador de eventos de una función HTTP detecta el evento `onRequest()` y admite dos argumentos específicos de HTTP: solicitud (`Request`) y respuesta (`Response`). El objeto `Request` da acceso a las propiedades de la solicitud HTTP que envió el cliente y el objeto `Response` proporciona una forma de enviar una respuesta al cliente.

```
exports.fecha = functions.https.onRequest((req, res) => {
   res.status(200).send(fechaFormateada);
});
```

Siempre termina una función HTTP con `send()`, `redirect()` o `end()`. De lo contrario, la función podría seguir ejecutándose y el sistema podría forzar su finalización. En el ejemplo, la función devuelve una fecha formateada en una variable llamada `fechaFormateada`, terminamos la función de la siguiente forma:

```
res.status(200).send(fechaFormateada);
```

Invoca la función a través de su URL. La URL es única. Incluye lo siguiente (en ese orden):

- la región en la que se implementa la función. Por ahora sólo se admite la región us-central1.
- el ID del proyecto de *Firebase*
- *cloudfunctions.net*
- el nombre de la función

Por ejemplo, la URL para invocar `fecha()` tiene el siguiente aspecto:

```
https://us-central1-<project-id>.cloudfunctions.net/fecha
```

Podrás ver la dirección de la función desde la consola de Firebase una vez que esté implementada. La podrás consultar desde el apartado *"Functions"*, en la lista de funciones o en las *"Estadísticas de uso detalladas"* de la función, en la pestaña *"Activador"*.

Veamos como llamar a una función pasándole parámetros. Vamos a utilizar dos métodos para realizar la llamada: `GET` y `POST`.

- `GET`: Realizaremos una llamada mediante el método `GET` añadiendo a la *URL* de la función, los parámetros y sus valores.

Para realizar una llamada `GET` bastará con crear un formulario HTML:

```
<form action="/url_funcion" method="get">
   Parámetro#1<input type="text" name="param1"><br>
   Parámetro#2<input type="text" name="param2"><br>
   <input type="submit" value="Enviar">
</form>
```

O podemos contruir directamente la *URL* para llamar a la función con los valores de los parámetros:

```
http://url_funcion?param1=valor1&param2=valor2&...&paramN=valorN
```

Añadiremos a la *URL* pares: nombre del parámetro y valor que sean necesarios. El primer parámetro irá precedido por un símbolo `?` y los siguientes por `&`. Indicaremos el valor del parámetro con el símbolo `=` seguido del valor.

- `POST`: Realizaremos una llamada mediante el método `POST` añadiendo a la llamada de la función. Pero estos no estarán visibles en la *URL* de la llamada.

  Bastará con crear un formulario HTML que utilice el método `POST`:

```
<form action="/url_funcion" method="post">
   Parámetro#1<input type="text" name="param1"><br>
   Parámetro#2<input type="text" name="param2"><br>
   <input type="submit" value="Enviar">
</form>
```

Veamos un ejemplo de como llamar a una función y pasarle un mensaje simple:

```
curl -X POST -H "Content-Type:application/json" -d '{"mensaje":"Hola
mundo"}' URL_FUNCION
```

El cuerpo de la solicitud se analiza de forma automática según el tipo de contenido y se llena en el cuerpo del objeto `request`. Por ejemplo:

Tipo de contenido	Cuerpo de la solicitud	Comportamiento
`application/json`	'{"nombre":"Pepe"}'	`request.body.nombre` equivale a "Pepe"
`application/octet-stream`	'texto'	`request.body` equivale a "6d792074657"
`text/plain`	'texto'	`request.body` equivale a "texto"
`application/x-www-form-urlencoded`	'nombre=Pepe'	`request.body.nombre` equivale a "Pepe"

Supongamos que se usa la siguiente solicitud para llamar a la función:

```
curl -X POST -H "Content-Type:application/json" -H "X-cabecera: 123"
URL_FUNCION?foo=baz -d '{"texto":"algún texto"}'
```

Los datos enviados se materializarían en:

Propiedad/Método	Valor
`request.method`	“POST”
`request.get('x-cabecera')`	“123”
`request.query.foo`	"baz"

`request.body.texto`	"algún texto"

Puedes conectar una función HTTP con *Firebase Hosting*. Las solicitudes en un sitio de *Firebase Hosting* se pueden dirigir a funciones HTTP específicas.

Ejercicio: *Crear un servicio Web con un activador HTTP.*

1. Abre el fichero index.js del directorio functions del proyecto "Eventos" con un editor de texto y añade la siguiente declaración al final del fichero:

```
const db = admin.firestore();
```

Para que *Firebase Functions* trabaje con *Firestore* tenemos utilizar una instancia de *Firestore*.

2. Añade el código de la función al final del fichero:

```
exports.mostrarEventos = functions.https.onRequest((req, res) => {
   var evento= req.query.evento;
   db.collection('eventos').doc(evento).get().then(doc => {
      if (doc.exists) {
         res.json(doc.data());
      } else {
         res.send("No se encuentra el evento "+evento+"");
      }
   }).catch(reason => {
      res.send("Error");
   })
})
```

Hemos implementado una función que para que recoge los valores de los parámetros mediante el método `GET`. Los valores de los parámetros los obtendremos con la propiedad `query` de `Request`. Conseguimos el valor del parámetro `evento` con `req.query.evento`. Consultamos los datos de Firestore mediante `db.collection('eventos').doc(evento).get()` y los devolvemos en formato *JSON* con `res.json`.

3. Desde la consola del sistema operativo ejecuta la siguiente instrucción situándote en la carpeta *functions* del proyecto:

```
firebase deploy --only functions
```

4. Accede a la consola de Firebase, comprueba que se ha subido la función y copia la dirección URL que se ha generado. La encontrarás debajo del nombre de la función.
5. Abre un navegador web, pega en la barra de direcciones la *URL* de la función y añade `?evento=fallas`.

   Observa que el resultado devuelto es la información en formato *JSON* del evento `"fallas"` guardada en *Firestore*. Además de información *JSON* o de otro tipo para ser tratada desde la aplicación de la que se realiza la llamada, las funciones *HTTP* son capaces de generar páginas dinámicas.

6. Añade la función `mostrarEventosHtml` al final del fichero:

```
exports.mostrarEventosHtml = functions.https.onRequest((req, res) => {
   var evento= req.query.evento;
   db.collection('eventos').doc(evento).get().then(doc => {
      if (doc.exists) {
         res.status(200).send(`<!doctype html>
           <head>
             <link rel="stylesheet"
                   href="https://fonts.googleapis.com/css?family=Ranga">
           </head>
           <body>
              <span style="font-family: 'Ranga',serif;
                                                    font-size:medium;">
                 El evento ${doc.data().evento} se realiza en
                 la ciudad de ${doc.data().ciudad} el dia
                 ${doc.data().fecha}.
              </span>
           </body>
           </html>`);
        } else {
           res.send("No se encuentra el evento "+evento+"");
        }
   }).catch(reason => {
      res.send("Error");
   })
})
```

Utilizamos el método `res.status(200).send` para devolver una página dinámica. La devolución es una cadena con el código de una página HTML. En nuestro caso, generamos un mensaje indicando que el evento, que hemos pasado como parámetro, se realiza en una ciudad en una determinada fecha. Los datos son extraidos de *Firestore*. Para incluir el valor de un campo de la base de datos en una cadena utilizamos `${doc.data().nombre_campo}`. Por ejemplo, para utilizar el valor del campo ciudad utilizaríamos `${doc.data().ciudad}`.

7. Desde la consola del sistema operativo ejecuta la siguiente instrucción situándote en la carpeta *functions* del proyecto:

```
firebase deploy –only functions
```

8. Abre un navegador web, pega en la barra de direcciones la *URL* de la función y añade `?evento=fallas`.

   Comprueba que aparece la frase: *“El evento Fallas se realiza en la ciudad de Valencia el dia 19/03/2017.”*.

Práctica: *Información del día del evento.*

Vamos a informar al usuario de la ciudad y día de un evento con la frase que hemos creado en el ejercicio anterior. En la actividad `EventosWeb` añade un menú Android o un botón Jquery Mobile, lo que prefieras, que al pulsarlo hará que se cargue la página web creada en el ejercicio anterior del evento que estamos viendo.

5.1.3.8. Activadores de Pub/Sub de Cloud

Pub/Sub de *Google Cloud* es un servicio que permite enviar mensajes entre aplicaciones. Una aplicación puede enviar mensajes a un tema y otras aplicaciones pueden suscribirse a dicho tema para recibirlos. Utiliza `functions.pubsub` para crear funciones que controlen eventos `Pub/Sub` de *Google Cloud*.

El evento disponible es:

Evento	Descripción
`onPublish`	Se envía un mensaje.

Cloud Functions admite el evento `Message`. Este evento se activa cada vez que un mensaje de *Pub/Sub* se envía a un tema específico. Debes especificar el nombre del tema que deseas que active la función y configurar el evento dentro del controlador de eventos `onPublish()`:

```
exports.funcion =
        functions.pubsub.topic('tema').onPublish((message, context) => {
  // Código a ejecutar
});
```

Puedes acceder a la carga útil del mensaje de *Pub/Sub* desde `message` como un objeto `Message`. En el caso de los mensajes con *JSON* en el cuerpo del mensaje *Pub/Sub*, el SDK de *Firebase* para *Cloud Functions* tiene una propiedad de ayuda que decodifica el mensaje. Por ejemplo, a continuación, se muestra un mensaje publicado con una carga útil *JSON* simple:

```
gcloud beta pubsub topics publish new_topic '{"name":"Xenia"}'
```

Puedes acceder a una carga de datos *JSON* como la anterior a través de la propiedad `json`:

```
let name = null;
try {
   name = message.json.name;
} catch (e) {
   console.error('El mensaje PubSub no está en JSON', e);
}
```

Las cargas útiles que no son *JSON* se incluyen en el mensaje de *Pub/Sub* como un `string` codificado en base 64 en `message`. Para leer un mensaje como el siguiente, debes decodificar el `string` codificado en base 64 como se muestra:

```
const messageBody = message.data ?
             Buffer.from(message.data, 'base64').toString() : null;
```

El mensaje de *Pub/Sub* se puede enviar con atributos de datos configurados en el comando de publicación. Por ejemplo, podrías publicar un mensaje con un atributo `name`:

```
gcloud beta pubsub topics publish new_topic --attribute name=Xenia
```

Puedes leer estos atributos desde `message.attributes`:

```
const name = message.attributes.name
```

Preguntas de repaso: *Firebase Functions.*

5.2. Enlaces dinámicos

5.2.1. Dynamic Links

Cuando te pasan un enlace a un video de YouTube, la forma habitual de abrirlo es a través del navegador Web. Sin embargo, es posible que tengas instalada la aplicación de YouTube, que te aportaría una mejor experiencia de usuario. Gracias a Dynamics Links vamos a poder definir URL asociadas a nuestra aplicación, de forma que el usuario pueda abrir contenido. En caso de no tenerla instalada se le puede sugerir que la instale, o si no está interesado, o estás en un sistema no Android, abrir el contenido desde la Web.

Firebase Dynamic Links son URLs inteligentes que cambian su comportamiento de forma dinámica para proporcionar la mejor experiencia en diferentes plataformas.

Un mismo enlace puede apuntar a contenidos diferentes según la plataforma en la que es abierto. Si un usuario abre un enlace dinámico y no tiene la aplicación instalada, se le puede solicitar que la instale; luego, después de la instalación, se iniciará la aplicación y podrá acceder al vínculo. Cuando un usuario abre un *Dynamic Links*, si aún no tiene instalada la aplicación, se le dirige a *Play Store* o *App Store* para que la instale (a menos que especifiques algo diferente), y luego se abre la aplicación. Entonces, podrás recuperar el vínculo que se le pasó a la aplicación.

Los enlaces dinámicos son duraderos y sobreviven a las instalaciones de la aplicación. Evita perder conversiones cuando los usuarios potenciales aún no tengan la aplicación instalada. Funcionan en iOS, Android y en aplicaciones web de escritorio y móvil.

Úsalos en campañas de comercialización y para compartir contenido a fin de saber exactamente qué campañas y qué contenido están impulsando el crecimiento. Los análisis en la consola de *Firebase* permiten comprender qué vínculos generan la instalación y el uso de la aplicación.

Usa *Dynamic Links* en promociones físicas, web, por correo electrónico, en redes sociales y de recomendación para aumentar la captación y retención de usuarios. Combina su uso con *Firebase Analytics*.

Funciona con o sin la aplicación instalada. Si es necesario que esté instalada para abrir un vínculo, la aplicación abre automáticamente el vínculo después de la instalación. Si es necesario actualizar la aplicación para abrir un vínculo. El vínculo se abrirá automáticamente después de actualizarse. Esta característica solo está disponible en Android.

Puedes llevar a los usuarios directamente a contenido vinculado dentro de la aplicación. Los vínculos se abren directamente en la aplicación, sin necesidad de diálogo de desambiguación ni rebote a través de un navegador.

Consulta un análisis básico de cada enlace en el panel de *Dynamic Links* de la consola de *Firebase* o uno avanzado usando *Firebase Analytics*.

Puedes crear un *Dynamic Link* desde la consola de Firebase o desde programación.

El uso de *Dynamic Link* es gratuito e ilimitado.

Veamos cómo crear un enlace dinámico en la consola de Firebase:

Ejercicio: *Crear un vínculo dinámico en Firebase.*

1. Abre el proyecto “Eventos” en la consola de *Firebase*.

   Si realizas un nuevo proyecto asegúrate de haber proporcionado la huella digital SHA1 de la aplicación Android en la página de configuración del proyecto.

2. Abre la página de *“Dynamic links”* en el menú de la barra lateral.

   En esta página podrás crear y ver tus enlaces creados, así como su tasa de clics.

3. Vamos a crear el subdominio sobre el que se ejecutarán los vínculos. Para crear el subdominio haz clic en el botón *“Comenzar”*.

4. Introduce en el nombre del subdominio el identificador de la aplicación en Firebase (*eventos-xxxx*), verifica que está disponible y finaliza el asistente.

   Dynamic Links se crean en el dominio *page.link*. Puedes personalizar el vínculo con el nombre que creas, pero es bastante común introducir el nombre de la aplicación o marca.

5. Para crear el primer vínculo dinámico pulsa sobre el botón *“Nuevo vínculo dinámico”*.

6. No vamos a personalizar la url corta que nos propone Google para el vínculo dinámico. Pulsa en el botón *“Siguiente”*.

7. Proporciona la información básica de un enlace dinámico:

   URL de vínculo directo: https://eventos-xxxx.firebaseapp.com/

   Nombre: Vinculo básico

Un vínculo dinámico es un vínculo directo a una aplicación. Funciona sin importar si está instalada. En los ordenadores de escritorio, se dirigirá a la URL de vínculo directo. Será el vínculo que abrirá tu aplicación. El vínculo debe ser una URL válida que utilice el esquema HTTPS o HTTP.

Nombre del vínculo dinámico, será al que harás referencia cuando realices el seguimiento de datos, como los clics en este vínculo

8. Pulsa en el botón *"Siguiente"*.
9. Nos pide que definamos el comportamiento en iOS. Dejaremos la opción por defecto, "Abrir el vínculo directo en un navegador".

   Existe la opción de abrir el vínculo en la aplicación iOS.
10. Pulsa en el botón *"Siguiente"*.
11. En la siguiente pantalla, tendremos que definir el comportamiento en Android. Marca la opción *"Abrir el vínculo directo en la app de Android"*.
12. Nos pedirá que elijamos el paquete de la aplicación que deseamos que abra el vínculo. Selecciona `org.example.eventos`.

    Al igual que en iOS tenemos la posibilidad de elegir como se va a comportar el vínculo cuando sea abierto por un dispositivo Android. En iOS hemos elegido que se abra en un navegador, por lo tanto, se abrirá la URL que hemos indicado en el vínculo directo, aunque la aplicación esté instalada en el dispositivo. Sin embargo, en Android hemos seleccionado la opción para que la aplicación sea la que abra el vínculo.
13. Elige la opción *"App instantánea o URL personalizada"* en *"Si no está instalada la app, envía el usuario a"*.

    Hemos indicado que la aplicación sea la que abra el vínculo. Pero tenemos que indicar como se va a comportar el vínculo si el usuario no tiene la aplicación instalada. Tenemos tres opciones.

    - **Página de Google Play para tu app:** envía al usuario a Google Play para que instale la aplicación.
    - **App instantánea o URL personalizada:** tenemos la posibilidad de enviar al usuario a una URL determinada o bien, si la aplicación se configuró con Apps instantáneas, este vínculo abrirá y ejecutará la aplicación nativa de forma instantánea, sin necesidad de realizar ninguna instalación.
14. Introduce la siguiente URL en el campo de "App instantánea o URL personalizada": https://www.androidcurso.com/
15. Haz clic en el botón *"Siguiente"*.
16. De forma opcional, podemos unir el vínculo a una campaña para poder seguirla y evaluar los resultados. No lo vamos a utilizar.
17. Pulsa en el botón *"Siguiente"*.

18. Puedes personalizar el vínculo para cuando se comparta mediante las redes sociales. Puedes introducir un título, una descripción y una imagen, que acompañarán al vínculo en las redes sociales. No lo vamos a utilizar.
19. Pulsa en *"Crear vínculo dinámico"*.
20. Copia la URL del vínculo que se ha creado y que la obtendrás en la columna URL de la lista de vínculos creados.
21. Comparte el vínculo y ábrelo en tu ordenador y en un dispositivo Android.

Una forma bastante fácil es auto enviarse un email a la misma cuenta de Google que se ejecuta en el dispositivo Android. A estas cuentas de correo, normalmente, puedes acceder desde el dispositivo Android y desde el ordenador. Mediante el envío del email, simulamos que el usuario ha recibido el vínculo como parte de una campaña de email marketing.

Abre el vínculo en tu ordenador desde el email que te has enviado. Comprueba que abre la URL de vínculo directo. Luego abre el vínculo desde el dispositivo Android donde tienes la aplicación *"Eventos"* instalada, comprueba que se abre la aplicación. Desinstala la aplicación en el dispositivo y vuelve a abrir el vínculo, observa que se abre la URL www.androidcurso.com. Si la aplicación no está instalada, podemos indicar que acción se va a realizar al abrirlo. Lo normal, es pedir al usuario que instale la aplicación redirigiéndolo a *Google Play* para que la instale.

Una de las características más importantes de los vínculos dinámicos es que puedes dirigir al usuario una parte concreta de la aplicación o mostrar un contenido concreto. Incluso si el usuario no tiene instalada la aplicación, cuando se la instala, el usuario será redirigido al contenido concreto dentro de la aplicación cuando la abra.

Por ejemplo, supongamos que enviamos una campaña de email sobre el evento "Fallas". Introducimos en el email un vínculo dinámico para que los usuarios que lo pulsen vean la información del evento en nuestra aplicación.

Vamos a modificar la aplicación para que esto se produzca y crearemos un vínculo en Firebase para que muestre un evento concreto.

Ejercicio: *Abrir un vínculo dinámico en una actividad.*

1. Añade el siguiente filtro en la declaración de la actividad `EventosDetalles` en el manifiesto:

```
<intent-filter>
   <action android:name="android.intent.action.VIEW"/>
   <category android:name="android.intent.category.DEFAULT"/>
   <category android:name="android.intent.category.BROWSABLE"/>
   <data android:host="eventos-xxxx.firebaseapp.com"
                                                android:scheme="https"/>
</intent-filter>
```

Sustituye `eventos-xxxx.firebaseapp.com` por la dirección de *Firebase Hosting* de la aplicación. Debemos declarar este filtro en cada una de las actividades que quieras que se abran desde un vínculo dinámico. La declaración de `action` y `category`, las tienes que realizar tal y como indica el paso del ejercicio. En la declaración de `data` es donde vamos a personalizar la declaración. Lo que va a recibir la aplicación va a ser la URL que se introduzca en la URL de vínculo directo. La URL declarada en nuestro ejemplo es la siguiente: `https://eventos-xxxx.firebaseapp.com`. Distinguimos el dominio (`eventos-xxxx.firebaseapp.com`) y el esquema (`https`).

2. Si has desinstalado la aplicación, vuelve a instalarla. Ejecuta la aplicación y ciérrala. Si no la has desinstalado, revisa que la aplicación se encuentre cerrada.

3. Abre el vínculo del ejercicio anterior en el dispositivo.

   Revisa que cuando se abre la aplicación no se abre la actividad inicial, si no que se abre la actividad `EventosDetalles`. Muestra la actividad en blanco, ya que no le hemos indicado que muestre ningún evento. Lo resolveremos más adelante.

   Podemos abrir distintas actividades según la URL que enviemos a la aplicación. En la declaración de cada actividad en el manifiesto, deberemos declarar filtros distintos. Por ejemplo, si tenemos una aplicación de cocina con recetas paso a paso y videos, podríamos declarar un filtro para el dominio `recetas.miaplicacion.com` para la actividad de las recetas y otro filtro `videos.miaplicacion.com` para la actividad que muestra los videos. También deberemos declara el esquema, es decir, si la llamada se va a hacer mediante HTTP o HTTPS. El parámetro data del filtro de recetas quedaría de la siguiente forma: `<data android:host="recetas.miaplicacion.com" android:scheme="http"/>`. Si las recetas recibieran URL HTTPS y HTTP, podríamos declarar más parámetros data en el mismo filtro. Añadiríamos el filtro del ejemplo pero con `android:scheme="https"`.

Ejercicio: *Abrir un vínculo dinámico con sobrecarga de datos.*

1. Crea un vínculo dinámico desde la consola de Firebase con las mismas opciones que el vínculo creado anteriormente, pero con los siguientes cambios.

   URL de vínculo directo: https://eventos-xxxx.firebaseapp.com?evento=fallas

   Nombre: Vinculo con sobrecarga

2. Una vez hayas creado el vínculo, reenvíatelo mediante email.

3. Sustituye en el método `onCreate` de la clase `EventoDetalles` la instrucción:

```
if (evento==null) evento = "";
```

   Por:

```
if (evento==null) {
    android.net.Uri url = getIntent().getData();
    evento= url.getQueryParameter("evento");
}
```

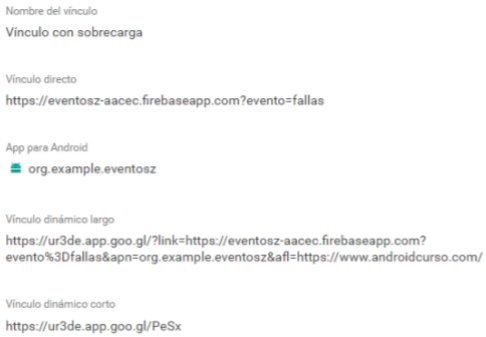
4. Ejecuta la aplicación en el dispositivo y ciérrala.
5. Pulsa sobre el vínculo “Vínculo con sobrecarga” que acabas de enviarte por email.

Comprueba que se abre la aplicación y se muestra directamente la información sobre el evento “fallas”. Si en usuario no tuviera instalada la aplicación, lo podríamos redirigir a Google Play para que la instalara. Cuando la terminara de instalar y la abriera, mostraría el evento “fallas”, ya que la instalación se realizó mediante un vínculo dinámico.

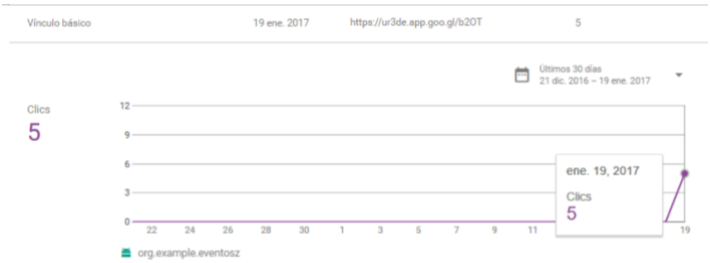
Hemos indicado que *Dynamic Link* se integra con *Firebase Analytics*. Por eso, sin realizar ninguna programación especial, podemos disponer de una estadística de clics sobre el enlace. Pulsa sobre un enlace en la lista que aparece en *Dynamic Links*.

Para cada vínculo de la lista, podemos ver la siguiente información utilizando su menú individual. Tenemos tres elementos en el menú:

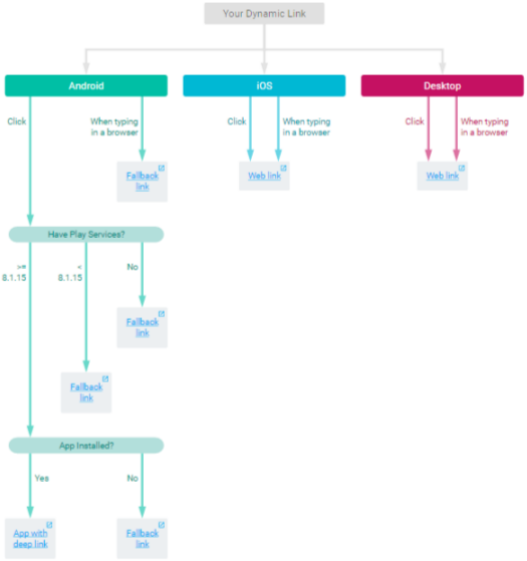
- **Detalles del vínculo:** Muestra la información del vínculo: nombre, enlace directo, la url del vínculo dinámico corto y largo, …



- **Estadísticas del vínculo:** Muestra los clics que se han hecho sobre el vínculo de forma temporal.



- **Flujo de vínculos:** Revisa el flujo del vínculo dependiendo del sistema operativo del dispositivo donde abra el vínculo el usuario, de la versión de Google Play Services, de si la aplicación está instalada, … Además, puedes verificar los links de cada caso.



Si observas los detalles del vínculo, verás que nos muestra un "Vínculo dinámico corto" (es el que hemos utilizado hasta ahora, el que hemos enviado por email) y "Vínculo dinámico largo". Es el vínculo dinámico largo el que procesa Android, pero es encapsulado en un vínculo corto.

Firebase ha encapsulado el vínculo: https://ur3de.app.goo.gl/?link=https://eventosz-aacec.firebaseapp.com?evento%3Dfallas&apn=org.example.eventosz&afl=https://www.androidcurso.com/ en el siguiente vínculo corto: https://ur3de.app.goo.gl/PeSx. También, podemos crear un Dynamic Link mediante programación. Crearíamos un vínculo largo. Debes construir una URL con la siguiente forma:

```
https://domain/?link=link&apn=package_name[&amv=minimum_version][&ad=1]
                [&al=android_link][&afl=fallback_link]
```

- *domain*: El dominio *Dynamic Links* del proyecto en *Firebase*.
- link: El vínculo que abrirá la aplicación. Puedes especificar cualquier URL compatible con la aplicación, como un vínculo al contenido o una URL que inicie una lógica específica. Este enlace debe ser una URL con formato correcto, debe estar correctamente codificado y debe usar un esquema HTTP o HTTPS.
- apn: El nombre del paquete de la aplicación Android que abrirá el vínculo. La aplicación debe estar conectada al proyecto de *Firebase*. Es obligatorio para que *Dynamic Link* abra la aplicación Android.

- `amv`: (opcional) La `versionCode` de la versión mínima de la aplicación que puede abrir el vínculo. Si la aplicación instalada es una versión anterior, se dirige al usuario a Play Store para que la actualice.
- `ad` (opcional) Declara que el *Dynamic Link* se está utilizando en un anuncio.
- `al`: (opcional) El vínculo que se debe abrir en Android. Este vínculo puede usar cualquier esquema y, si se especifica, tiene prioridad sobre el parámetro link en Android.
- `afl`: (opcional) El vínculo que se debe abrir cuando no esté instalada la app. Se usa para especificar que haga algo diferente de instalar la aplicación desde Play Store cuando no esté instalada, como abrir la versión web móvil del contenido o mostrar una página promocional para tu app.
- `utm_source, utm_medium, utm_campaign, utm_term, utm_content, gclid`: (opcional) Parámetros de una campaña personalizada. Estos parámetros se pasan a vínculos profundos para que la aplicación registre resultados para su análisis.

Hasta el momento hemos visto como enviar vínculos dinámicos desde la consola de Firebase que pueden ser abiertos por la aplicación para mostrar un contenido específico. También puedes crear vínculos dinámicos mediante programación que podrán ser creados desde fuentes externas a la consola de Firebase, por ejemplo, un servidor propio.

Práctica: *Campaña de descuento en evento.*

Imaginemos que realizamos una campaña de marketing por email en la que se ofrecen descuentos para asistir a un evento. Enviaremos un email con un enlace dinámico para el evento *"fallas"* con un descuento del 25 %. Al pulsar sobre el enlace se abrirá la actividad `EventosWeb` de la aplicación, donde se mostrará la información del evento y el descuento conseguido. Deberás mostrar el descuento en la aplicación web de la actividad.

5.2.2. Invitaciones con Firebase Dynamic Links

Los vínculos dinámicos otorgan un extra a nuestra aplicación para captar y retener a usuarios. Pero, está demostrado, que la mejor forma de que un usuario instale una aplicación es mediante una recomendación de un familiar o conocido del usuario. Podiamos potenciar esta alternativa de promoción con *Firebase Invites*, activo hasta enero de 2020. *Firebase* ha integrado esta funcionalidad en *Firebase Dynamic Links*.

El boca a boca es una de las formas más eficaces de lograr que los usuarios instalen tu aplicación. Aprovecha a los usuarios de la aplicación como sus principales promotores.

Para crear vínculos de invitación: crea un Dynamic Link que tus usuarios puedan compartir con sus amigos. También puedes incluir parámetros específicos en el enlace, como metadatos de redes sociales que permitirán personalizar el aspecto de la URL que se comparte, si tus usuarios comparten tu app mediante estas plataformas.

Procedamos a modificar la aplicación "Eventos" para que permita enviar invitaciones y procesar las invitaciones recibidas.

Ejercicio: *Enviar invitaciones desde una aplicación Android.*

1. Abre la consola de Firebase y crea un nuevo vínculo seleccionando *"Dynamic links"* en el menú de la barra lateral.
2. No vamos a personalizar la url corta que nos propone Google para el vínculo dinámico. Pulsa en el botón *"Siguiente"*.
3. Proporciona la información básica de un enlace dinámico:

   URL de vínculo directo: https://invitacion.eventos-xxxx.firebaseapp.com/

   Nombre: Invitacion

   Cambia `eventos-xxxx` por el dominio de tu aplicación en *Firebase*.
4. Pulsa en el botón *"Siguiente"*.
5. Nos pide que definamos el comportamiento en iOS. Dejaremos la opción por defecto, "Abrir el vínculo directo en un navegador". Pulsa en el botón *"Siguiente"*.
6. En la siguiente pantalla, tendremos que definir el comportamiento en Android. Marca la opción *"Abrir el vínculo directo en la app de Android"*.
7. Elige el paquete `org.example.eventos`.
8. Elige la opción *"App instantánea o URL personalizada"* en *"Si no está instalada la app, envía el usuario a"*. Introduce la siguiente URL en el campo de "App instantánea o URL personalizada": https://www.androidcurso.com/
9. Haz clic en el botón *"Siguiente"*.
10. De forma opcional, podemos unir el vínculo a una campaña para poder seguirla y evaluar los resultados. No lo vamos a utilizar. Pulsa en el botón *"Siguiente"*.
11. Puedes personalizar el vínculo para cuando se comparta mediante las redes sociales. No lo vamos a utilizar.
12. Pulsa en *"Crear vínculo dinámico"*.

    Vamos a distinguir entre los vínculos dinámicos enviados desde la consola de Firebase, donde el enlace contiene la información de un evento y las invitaciones, en las que simplemente vamos a precisar que se abra la aplicación. En el primer caso, se abrirá la actividad `EventoDetalles` y en el caso de las

invitaciones se abrirá la actividad `MainActivity`. Para ello tendremos que especificar dos filtros diferentes a definir en la declaración de las actividades en el manifiesto. Distinguiremos dos `host` diferentes en el manifiesto, para `EventoDetalles` será `eventos-xxx.firebaseapp.com` y para `MainActivity` será `invitacion.eventos-xxxx.firebaseapp.com`.

13. Inserta el siguiente filtro en la declaración de la actividad `MainActivity` en el manifiesto:

```
<intent-filter>
   <action android:name="android.intent.action.VIEW"/>
   <category android:name="android.intent.category.DEFAULT"/>
   <category android:name="android.intent.category.BROWSABLE"/>
   <data android:host="invitacion.eventos-xxxx.firebaseapp.com"
         android:scheme="http"/>
   <data android:host="invitacion.eventos-xxxx.firebaseapp.com"
         android:scheme="https"/>
</intent-filter>
```

Cambia `eventos-xxxx` por el dominio de tu aplicación en *Firebase*.

14. Inserta el siguiente elemento de menú en `main_menu`, delante del elemento `action_temas`:

```
<item
   android:id="@+id/action_invitar"
   android:orderInCategory="50"
   android:title="Invitar"
   app:showAsAction="always" />
```

15. Inserta dentro del método `onOptionsItemSelected` de la clase `MainActivity`, delante de `return super.onOptionsItemSelected(item);` :

```
if (id == R.id.action_invitar){
   invitar();
}
```

16. Declara la siguiente variable con el valor de la URL del vínculo dinámico que hemos creado anteriormente:

```
private String dlInvitacion = "https://eventos-xxxx.page.link/xxxx";
```

17. Añade la final de la clase `MainActivity`:

```
private void invitar() {
   Intent sendIntent = new Intent();
   sendIntent.setAction(Intent.ACTION_SEND);
   sendIntent.putExtra(Intent.EXTRA_TEXT, "No te pierdas nunca más un
                              evento interesante.:\n\n" + dlInvitacion);
   sendIntent.putExtra(Intent.EXTRA_SUBJECT, "Envio de una invitación
                                                   a la app Eventos");
   sendIntent.setType("text/plain");
```

```
    startActivity(Intent.createChooser(sendIntent, null));
}
```

18. Ejecuta la aplicación.

Comprueba que puedes enviar una invitación desde el menú “INVITAR” en la pantalla inicial. Al pulsar sobre el botón “INVITAR” aparecerá una nueva pantalla donde podrás seleccionar la aplicación con la cuál enviar la invitación y envíala. Verifica que se ha recibido la invitación. Comprueba que al pulsar sobre el vínculo se abre la aplicación.

En resumen, hemos conseguido enviar un vínculo dinámico con el propósito de dar a conocer nuestra aplicación de una forma más eficiente.

Un usuario puede que envíe una invitación más fácilmente si tiene una motivación. Por ejemplo, un descuento. Para ello necesitas pasar la información del descuento en la URL del enlace dinámico, al igual que hemos hecho con el enlace que hemos creado en la consola de Firebase que nos permitía abrir un evento en concreto. Vamos a modificar la aplicación para que el usuario pueda enviar una invitación a un evento con un descuento asociado. En el anterior ejercicio hemos creado el enlace dinámico desde la consola de Firebase. En este ejercicio, si seguimos el mismo procedimiento, tendríamos que crear un enlace para cada evento. Pero tenemos la opción de crear enlaces dinámicos de forma programática desde la aplicación. Esta es la opción que vamos a escoger.

Ejercicio: *Enviar invitaciones con sobrecarga de datos.*

1. Inserta el siguiente filtro en la declaración de la actividad `EventoDetalles` en el manifiesto:

```
<intent-filter>
   <action android:name="android.intent.action.VIEW"/>
   <category android:name="android.intent.category.DEFAULT"/>
   <category android:name="android.intent.category.BROWSABLE"/>
   <data android:host="descuento.eventos-xxxx.firebaseapp.com"
         android:scheme="http"/>
   <data android:host="descuento.eventos-xxxx.firebaseapp.com"
         android:scheme="https"/>
</intent-filter>
```

Cambia `eventos-xxxx` por el dominio de tu aplicación en *Firebase*.

2. Inserta el siguiente elemento al final del menú en `main_detalle`:

```
<item
   android:id="@+id/action_descuento"
   android:orderInCategory="105"
   android:title="Descuento"
   app:showAsAction="always" />
```

3. Inserta un nuevo caso en la instrucción `switch` del método `onOptionsItemSelected` de la clase `EventoDetalles`:

```
case R.id.action_descuento:
   if (id == R.id.action_descuento){
      invitarDescuento();
   }
   break;
```

4. Añade la final de la clase `EventoDetalles`:

```
private void invitarDescuento(){
   String dlInvitacionDescuento;
   dlInvitacionDescuento = "https://eventos-xxxx.page.link/";       //(1)
   dlInvitacionDescuento = dlInvitacionDescuento +
            "?link=https://descuento.eventos-xxxx.firebaseapp.com”; //(2)
   dlInvitacionDescuento = dlInvitacionDescuento +
                              "?descuento%3D25%26evento%3D"+evento; //(3)
   dlInvitacionDescuento = dlInvitacionDescuento +
                                    "&apn=org.example.eventos-xxxx"; //(4)
   dlInvitacionDescuento = dlInvitacionDescuento +
                              "&afl=https://www.androidcurso.com/"; //(5)
   Intent sendIntent = new Intent();
   sendIntent.setAction(Intent.ACTION_SEND);
   sendIntent.putExtra(Intent.EXTRA_TEXT,
          "Eventos con decuento del 15%:\n\n" + dlInvitacionDescuento);
   sendIntent.putExtra(Intent.EXTRA_SUBJECT, "Eventos con descuento");
   sendIntent.setType("text/plain");
   startActivity(Intent.createChooser(sendIntent, null));
}
```

5. Ejecuta la aplicación. Entra en los detalles de un evento, pulsa sobre el menú “Descuento” y envía el enlace.
6. Cierra la aplicación.
7. Pulsa sobre el enlace que has recibido en la aplicación que has enviado el enlace.

Comprueba que al pulsar sobre el enlace y abrir la aplicación, se muestran los detalles del evento y un mensaje que indica que se ha conseguido un 15% de descuento gracias al parámetro “descuento” del enlace. Faltaría procesar la invitación con el descuento. Es decir, que el usuario pueda aplicarse el descuento en la próxima compra y que no se pueda beneficiar más de una vez.

Hemos creado un vínculo dinámico de forma programática. Vamos a analizar como lo hemos hecho. Creamos el vínculo como una cadena en la variable dlInvitacionDescuento. Hemos dividido la composición del enlace en 5 partes identificadas mediante un número al final:

(1) `domain`: dominio *Dynamic Links* del proyecto en *Firebase*.

(2) `link`: vínculo que abrirá la aplicación sin parámetros.

(3) `link`: parámetros del vínculo: evento y descuento.

(4) `apn`: nombre del paquete de la aplicación Android que abrirá el vínculo.

(5) `afl`: vínculo que se debe abrir cuando no esté instalada la aplicación.

Cuando un usuario recibe una invitación, si aún no ha instalado la aplicación, podrá hacerlo desde Google Play Store. Una vez la haya instalado, o si ya estaba instalada, se inicia y recibe la URL de su contenido.

Preguntas de repaso: *Enlaces dinámicos.*

5.3. Configuración remota con Firebase

Firebase Remote Config es un servicio en la nube que te permite cambiar el aspecto y el comportamiento de una aplicación sin que los usuarios necesiten descargarla y actualizarla. Para usar *Remote Config*, debes crear valores predeterminados en la aplicación. Luego, puedes usar la consola de *Firebase* para cambiar estos valores predeterminados para todos o para unos segmentos de usuarios. La aplicación busca frecuentemente actualizaciones y las aplica con un impacto insignificante en el rendimiento.

Sus funciones clave son:

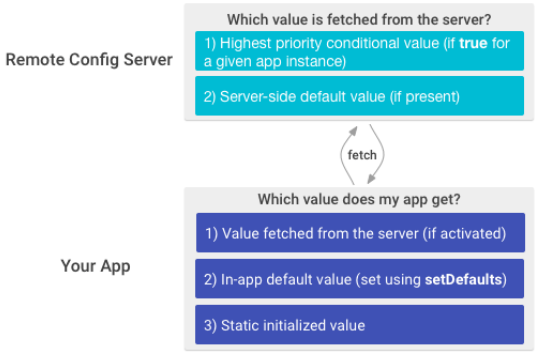
- **Implementación rápida de modificaciones:** Puedes realizar modificaciones en el aspecto y el comportamiento cambiando valores desde el servidor.
- **Personalización para segmentos de usuarios:** Puedes usar *Remote Config* para proporcionar variaciones en la aplicación para diferentes segmentos de usuarios a través de la versión de la aplicación, de *Firebase Analytics*, del idioma y más.
- **Ejecuta pruebas A/B:** Puedes usar un porcentaje aleatorio de usuarios con *Firebase Analytics* para realizar pruebas A/B para mejoras. De este modo puedes validarlas antes de implementarlas en todos los usuarios.

Remote Config realiza las tareas de obtención y almacenamiento en caché de valores y otorga el control sobre cuándo se activan los valores nuevos. Esto permite controlar cualquier modificación.

Los valores predeterminados se pueden obtener desde la aplicación o desde el servidor. Los métodos `get` de la biblioteca *Remote Config* proporcionan un único punto de acceso a los valores. La aplicación obtiene los valores del servidor usando la misma lógica que usa para obtener valores predeterminados de la aplicación.

Usa *Firebase console* para crear parámetros. Utiliza el mismo nombre en el servidor que en la aplicación. Para cada parámetro, puedes configurar un valor en el servidor para anular el valor en la aplicación. También puedes crear valores condicionales para instancias de la aplicación que cumplen ciertas condiciones.

El siguiente gráfico muestra cómo se priorizan los valores en el servidor y en la aplicación:



Los pasos para la implementación de *Remote Config* son:

- **Desarrolla la aplicación usando parámetros:** Define qué aspectos del comportamiento y aspecto de la aplicación quieres poder cambiar usando *Remote Config* y tradúcelos en parámetros.
- **Configura los valores de los parámetros predeterminados:** Utiliza `setDefaults()` para configurar los valores predeterminados.
- **Agrega la lógica para obtener, activar y conseguir los valores de los parámetros:** La aplicación puede obtener los valores de los parámetros de manera segura y eficiente desde el servidor. No debes preocuparte por cuál es el mejor momento para obtener los valores o incluso si existe algún valor en el servidor. La aplicación usa métodos `get` para obtener el valor de un parámetro, de forma similar a como se realiza la lectura del valor de una variable local.
- **Configura los valores de los parámetros condicionales y predeterminados del servidor:** Puedes definir valores en *Firebase console* para cambiar los valores predeterminados en la aplicación. Puedes hacer esto antes o después de lanzar la aplicación, porque los mismos métodos `get` acceden a valores predeterminados en la aplicación y a valores obtenidos desde el servidor.

Ten en cuenta las siguientes políticas:

- No uses *Remote Config* para realizar las actualizaciones que requieren de una autorización del usuario. Esto puede hacer que la aplicación se perciba como poco confiable.

- No almacenes datos confidenciales en claves de parámetros de *Remote Config* o valores de parámetros. Es posible decodificar cualquier clave o valor de parámetro almacenado en la configuración de *Remote Config*.
- No intentes eludir los requisitos de la plataforma de destino de la aplicación usando *Remote Config*.

Cuando usas *Remote Config*, defines uno o más parámetros y proporcionas valores predeterminados en la aplicación. Puedes anular los valores predeterminados en la aplicación mediante la definición de los valores en el servidor usando la *Firebase console*.

Al usar la *Firebase console*, puedes dar nuevos valores a los parámetros, así como establecer condiciones.

Se utiliza una condición para que se apliquen a un grupo de instancias de la aplicación. Las condiciones están compuestas de una o más reglas que debe cumplir una instancia. Un valor condicional consiste en una condición y un valor que se otorga a las instancias que cumplen esa condición. Un parámetro puede tener múltiples valores condicionales que usan condiciones diferentes. Los parámetros pueden compartir condiciones dentro de un proyecto.

Los valores de parámetros del servidor se obtienen de acuerdo a la siguiente lista de prioridades:

1. Primero, se aplican los valores condicionales, si alguna de las condiciones se cumple. Si se cumplen múltiples condiciones, la que se muestra más arriba en *Firebase console* tiene prioridad. Puedes cambiar la prioridad de las condiciones arrastrando y soltando las condiciones en la pestaña “Condiciones”.
2. Si no hay o no se cumple ninguna condición, se proporciona el valor definido en el servidor. Si un parámetro no existe en el servidor o si el valor predeterminado está configurado en “Ningún” valor, entonces no se proporciona ningún valor.

Los valores de parámetros se obtienen a través de métodos `get` de acuerdo a la siguiente lista de prioridades:

1. Si un valor se obtuvo desde el servidor y luego se activó, la aplicación usa este valor. Los valores de parámetros activados son persistentes.
2. Si no se otorgó ningún valor desde el servidor o si no se activaron valores obtenidos desde el servidor, se usa el valor predeterminado en la aplicación.
3. Si no se configuró ningún valor predeterminado en la aplicación, se usa un valor tipo estático (como `0` para `int` y `false` para `boolean`).

Se admiten los siguientes tipos de condiciones:

- **Usuario en un percentil aleatorio:** Usa este campo para aplicar un cambio un porcentaje de instancias aleatorias que cumplan una cierta condición. Usa los operadores <= y > para segmentar usuarios en grupos que no se superponen. A cada instancia de la aplicación se le asigna un número entero o fraccionario aleatorio dentro de un proyecto, de modo que puedes usar una regla de este tipo para abordar a las mismas instancias de la

misma manera. Por ejemplo, para crear dos condiciones relacionadas y que cada una se aplique a un 0,05% no superpuesto de la base de usuarios, podrías tener una condición que incluya una regla de <= 0,05% y otra condición que incluya una regla de > 0,05% y <= 0,10%.

- **Tipo de SO**: Puedes aplicar una condición para las instancias que usen un determinado tipo de SO.
- **País/región de dispositivo:** Esta condición se cumple, si la instancia se encuentra en cualquiera de las regiones o países que se encuentran en la lista.
- **ID de la aplicación:** Selecciona una Id de la lista de aplicaciones asociadas con el proyecto de Firebase. Para aplicaciones Android, tiene que coincidir con `android:versionName` de la aplicación.
- **Versión de la aplicación:** Introduce un valor para especificar una versión concreta (o versiones relacionadas). Antes de usar esta condición, debes usar una regla de Id. de la aplicación para seleccionar una aplicación que esté asociada con tu proyecto Firebase. Tiene que coincidir con `android:versionCode`. Puedes proporcionar una lista de valores separados por comas para actuar sobre varias versiones.
- **Idioma del dispositivo:** Esta condición se cumple en las instancias cuyos dispositivos usen uno de los lenguajes que se encuentra en la lista.
- **Propiedad del usuario:** Selecciona una lista de usuarios usando Firebase Analytics. Esta regla requiere el ID de la aplicación en *Firebase*. Debido a que muchos usuarios Analytics se definen por eventos o por propiedades del usuario, que pueden estar basadas en acciones, puede tomar un tiempo que una condición tenga efecto.

Vamos a usar *Remote Config* en nuestra aplicación *Eventos*:

Ejercicio: *Configuración remota en Eventos.*

1. Agrega la dependencia para *Remote Config* en el archivo *build.graddle [Module:app]*:

```
implementation 'com.google.firebase:firebase-config:17.0.0'
```

2. Declara el objeto *Remote Config* y las variables que nos permitirán cambiar el aspecto y comportamiento de la aplicación en la clase `Comun`:

```
static FirebaseRemoteConfig mFirebaseRemoteConfig;
static String colorFondo;
static Boolean acercaDe;
```

3. Inicializa el objeto *Remote Config* y declara las siguientes variables añadiendo el siguiente código al final del método `onCreate` en la clase `MainActivity`:

```
mFirebaseRemoteConfig = FirebaseRemoteConfig.getInstance();
FirebaseRemoteConfigSettings configSettings =
                              new FirebaseRemoteConfigSettings
                              .Builder()
                              .setMiniumFetchIntervalInSeconds(3600)
                              .build();
mFirebaseRemoteConfig.setConfigSettingsAsync(configSettings);
```

`FirebaseRemoteConfig.getInstance()` se usa para almacenar los valores en la aplicación, obtener los valores actualizados desde el servidor y controlar cuando estarán disponibles los valores obtenidos en la aplicación. Mediante `FirebaseRemoteConfigSettings` configuramos los ajustes.

Establemos el intervalo en segundos para la actualización de los datos mediante `setMiniumFetchIntervalInSeconds`. *Remote Config* almacena los valores en la caché del dispositivo después de la primera solicitud con éxito. De forma predeterminada, la caché caduca después de 12 horas, puedes cambiar la caducidad predeterminada, indicando la caducidad deseada en `setMiniumFetchIntervalInSeconds`. Si los valores están en la caché un mayor tiempo que la caducidad deseada, *Remote Config* va a obtener los valores desde el servidor. Si la aplicación solicita valores actualizados varias veces antes del tiempo mínimo, las solicitudes se regulan y se le proporciona un valor en caché.

4. Si no existe, crea una carpeta `xml` en recursos y añade un archivo llamado `remote_config_default.xml` con el siguiente contenido:

```
<?xml version="1.0" encoding="utf-8"?>
<defaultsMap>
   <entry>
      <key>color_fondo</key>
      <value>WHITE</value>
   </entry>
   <entry>
      <key>acerca_de</key>
      <value>true</value>
   </entry>
</defaultsMap>
```

El fichero xml contiene un mapa de entradas de pares clave/valor. Cada entrada determinará el valor por defecto de un parámetro para la aplicación si no lo obtiene del servidor.

5. Obtiene los valores predeterminados de la aplicación desde el fichero xml añadiendo al final del método `onCreate` de la clase `MainActivity` con la siguiente instrucción:

```
mFirebaseRemoteConfig. setDefaultsAsync(R.xml.remote_config_default);
```

6. Copia al final del método `onCreate` de la clase `MainActivity`:

```
mFirebaseRemoteConfig.fetchAndActivate()
   .addOnCompleteListener(this, new OnCompleteListener<Boolean>() {
      @Override
      public void onComplete(@NonNull Task<Boolean> task) {
         if (task.isSuccessful()) {
            getColorFondo();
            getAcercaDe();
         } else {
            Toast.makeText(MainActivity.this, "Remote config error.",
                                          Toast.LENGTH_SHORT).show();
         }
      }
   });
```

Mediante el objeto `FirebaseRemoteConfig` y su método `fetchAndActivate` actualizamos el valor de las entradas. Esto no significa que las variables que usas estas entradas posean el nuevo valor, sino que el valor es almacenado actualizado en caché para ser asignado a las variables. Puedes acceder a los valores llamando a uno de los métodos disponibles `get<type>` (por ejemplo, `getString`).

Durante el desarrollo de la aplicación, probablemente desees actualizar la caché con mucha frecuencia para que te permita iterar rápidamente cuando desarrollas y pruebas la aplicación. Puedes establecer temporalmente un valor bajo a `setMiniumFetchIntervalInSeconds` del objeto `FirebaseRemoteConfigSettings`.

7. Añade al final de la clase `MainActivity` el siguiente código:

```
private void getColorFondo() {
   colorFondo = mFirebaseRemoteConfig.getString("color_fondo");
}
private void getAcercaDe() {
   acercaDe = mFirebaseRemoteConfig.getBoolean("acerca_de");
}
```

8. Para cambiar el color de fondo de las páginas del `WebView` en `EventosWeb`, añade al final del fichero `funciones.js` la siguiente función y actualiza el proyecto en *Firebase Hosting*:

```
function colorFondo(color){
   document.getElementById("pagina1").style.background= color;
   document.getElementById("pagina2").style.background= color;
}
```

9. Añade delante de `return true;` en el método `onCreateOptionsMenu` en la clase `EventoDetalles` el siguiente código para mostrar u ocultar el menú “Acerca de”:

```
if (!acercaDe) {
   menu.removeItem(R.id.action_acercaDe);
}
```

10. Sustituye el método `onPageFinished` de la clase `EventosWeb` por el siguiente:

```
@Override
public void onPageFinished(WebView view, String url) {
   dialogo.dismiss();
   navegador.loadUrl("javascript:colorFondo(\""+colorFondo+"\")");
   navegador.loadUrl("javascript:muestraEvento(\""+evento+"\");");
}
```

    Nos ha faltado programar el color de fondo. Para ello haremos una llamada a la función `colorFondo` con el color que queremos que se muestre.

11. Ejecuta la aplicación.

    Comprueba que la aplicación se ejecuta correctamente. Como todavía no hemos definido ni los parámetros ni sus valores en la consola de *Firebase*, la aplicación coge por defecto los valores que hemos definido en `remote_config_default.xml`. Vamos a dar de alta los parámetros en la consola de *Firebase* y a comprobar cómo cambia el comportamiento y aspecto de nuestra aplicación.

12. En la consola de *Firebase*, seleciona el proyecto “Eventos” y accede al menú *Remote Config*.

13. Pulsa sobre el botón *“AGREGA TU PRIMER PARÁMETRO”*.

14. Introduce la clave “acerca_de” con el valor “false”.

15. Pulsa en “Agregar parámetro” para que el parámetro sea accesible desde la aplicación.

16. Sal completamente de la aplicación y sin volverla a instalar, vuélvela a ejecutar.

    Comprueba que ya no aparece del menú “Acerca de” cuando accedes a los detalles de un evento. Puede que el cambio tarde algún tiempo. Baja el tiempo en `setMinimumFetchIntervalInSeconds` si tarda demasiado.

17. Cambia el valor de la clave “acerca_de” con el valor “true”.

18. Pulsa en “Actualizar” para que los cambios sean visibles para la aplicación.

19. Vuelve a cerrar completamente y ejecutar la aplicación.

    Comprueba que ahora sí que aparece el menú “Acerca de” que no aparecía anteriormente.

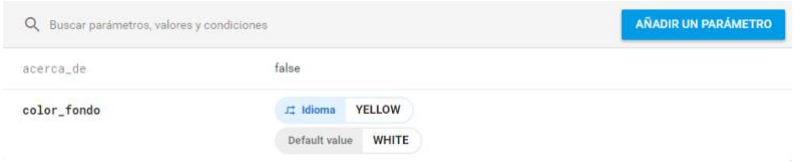
    Acabamos de crear un parámetro en Firebase console y le hemos asignado un valor. También hemos visto como cuando cambiamos el valor del parámetro, ha cambiado el comportamiento de la aplicación, mostrando y ocultando el menú “Acerca de”.

Tienes que tener en cuenta que los valores de los parámetros son leídos del servidor cuando se inicia la aplicación y haya expirado el tiempo que indicamos en `setMinimumFetchIntervalInSeconds`.

Remote Config no solamente acepta pares parámetro/valor, si no que podemos definir condiciones para que cambie el valor de un parámetro.

20. En la consola de *Firebase* pulsa el botón *"AÑADIR UN PARÁMETRO"* con el nombre *"color_fondo"*.
21. Pulsa sobre el texto "Agregar valor de condición" y en el diálogo que se abre pulsa sobre el texto "Definir una nueva condición" para añadir una condición.
22. Introduce "Idioma" en el nombre de la condición.
23. En "Se aplica si …", selecciona "Idioma del dispositivo", selecciona "afar(aa)" y pulsa sobre el botón "CREAR CONDICIÓN".

    Ya hemos creado nuestra primera condición. Se va a cumplir si la aplicación se encuentra instalada en un dispositivo que utilice el idioma "afar(aa)" como idioma. Ahora tenemos que definir el valor que tendrá el parámetro si se cumple la condición y que valor tendrá por defecto.
24. Introduce el valor "YELLOW" en "Valor de Idioma" y "WHITE" como valor por defecto.
25. Pulsa en "AÑADIR PARÁMETRO" para terminar con el alta.



26. Pulsa el botón "Agregar parámetro" para que se haga efectivo el uso del parámetro.
27. Si tienes abierta la aplicación, ciérrala y ejecuta la aplicación.

    Comprueba que el fondo del `WebView` de nuestra aplicación es blanco. A menos que el idioma del dispositivo sea "afar(aa)".
28. Edita el parámetro "color_fondo" y crea una nueva condición llamada "Idioma_Dispositivo" con el idioma del dispositivo en el que estás ejecutando la aplicación. El valor del parámetro para esta condición será "AZURE".
29. Recuerda pulsar en "Agregar parámetro" para que sean visibles para la aplicación.
30. Cierra y ejecuta la aplicación.

    Comprueba que el color del `WebView` ahora es azul.

Hemos visto cómo cambiar el comportamiento y aspecto de una aplicación sin tener que modificar la programación y sin que los usuarios tengan que actualizar la aplicación.

Práctica: *Configuración remota según país.*

Crea una pantalla de bienvenida (splash screen) que se mostrará al iniciar la aplicación. Se le dará la bienvenida al usuario a la aplicación y se le advertirá de que la aplicación mostrará eventos de España. Si el usuario se encuentra en España mostrará el mensaje “Esta aplicación muestra información sobre eventos” y si el usuario se encuentra en otro país, advertirá “Esta aplicación muestra eventos de España”. Prográmalo utilizando la configuración remota de Firebase.

Preguntas de repaso: *Configuración remota.*

5.4. Firebase In-App Messaging

5.4.1. Introducción

Firebase In-App Messaging es un servicio que permite involucrar a los usuarios activos de la aplicación mediante mensajes contextuales. Estos mensajes se denominan campañas.

Mediante el uso campañas vamos a animar a los usuarios a usar funciones claves de la aplicación. Por ejemplo: enviar un mensaje para se compre un determinado artículo, vean un video, … Los mensajes pueden personalizarse para que me muestren como tarjetas, modales, solo imágenes o banners. Se configuran activadores para que los mensajes aparezcan exactamente cuando sea oportuno.

Sus funciones más importantes son:

- **Enviar mensajes revelantes y atractivos**: *Firebase In-App Messaging* envía mensajes cuando se necesitan, cuando los usuarios están en la aplicación. Es decir: promociona la venta de un producto cuando esté mirando en la tienda, resalta una nueva característica de la aplicación, …
- **Dirige los mensajes por audiencia y comportamiento:** *Firebase In-App Messaging* trabaja juntamente con Analytics y Predictions, lo que permite utilizar herramientas para entregar los mensajes a unos determinados usuarios.
- **Crea alertas flexibles y personalizadas:** Puedes personalizar el estilo, la apariencia, los activadores y el contenido de los mensajes solamente haciendo clics.

5.4.2. Características

Veamos las principales características de *Firebase In-App*:

Diseño

Mediante plantillas puedes diseñar interfaces limpias y atractivas al usuario. Las plantillas disponibles son:

PLANTILLA	DESCRIPCIÓN
Tarjeta	Mensaje estructurado con dos botones de acción. Ofrece a los usuarios una opción.
Modal	Mensaje flexible de diálogo con un botón de acción. Sólo requiere el título del mensaje.
Solo imagen	Sube un diseño personalizado. Incorporación fácil de estética.
Banner	Mensaje de notificación. No ocupa mucho espacio en pantalla.

Destinos

Puedes dirigir cada campaña a determinado público en función de su comportamiento, idioma, compromiso, …

Programación

Los mensajes se muestran cuando los usuarios utilizan la aplicación y se activan ciertos eventos. Esto garantiza que sean revelantes y contextuales.

Estadísticas

Habilita eventos de conversión de *Analytics* para rastrear las interacciones de los usuarios con los mensajes. Permitirá revelar detalles importantes sobre las preferencias y satisfacción de los usuarios con la aplicación.

5.4.3. Campañas

El propósito de *Firebase In-App Messaging* es enviar mensajes relevantes a la aplicación cliente. Estos mensajes se llaman campañas.

Para usar *In-App Messaging* en *Android* necesitas:

- Añadir la aplicación a *Firebase* mediante *Firebase Console*
- Incluir *In-App Messaging SDK* en el *Gradle* de la aplicación.
- Conformar y enviar los mensajes desde *Firebase Console*.

Para ahorrar recursos, *Firebase In-App Messaging* recupera mensajes del servidor una vez al día. Esto dificulta el desarrollo, ya que necesitamos realizar pruebas. Firebase permite especificar un dispositivo de prueba que muestra mensajes a demanda. Vamos a ver como configurar el dispositivo de prueba:

Ejercicio: *Obtener el id de dispositivo de test.*

1. Agrega las siguientes dependencias en el archivo *build.graddle [Module:app]*:

```
implementation
            'com.google.firebase:firebase-inappmessaging-display:20.1.2'
implementation 'com.google.firebase:firebase-analytics:17.2.3'
```

Nota: Necesitarás actualizar bastantes librerías del proyecto ***"Eventos".*** Ten en cuenta que al hacerlo, puede que necesites actualizar algún método para que todo siga funcionando.

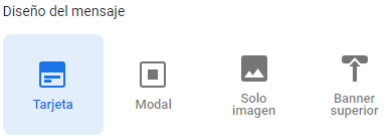
2. Ejecuta la aplicación.
3. Busca en el LogCat el siguiente registro en el nivel de "Info":

```
I/FIAM.Headless: Starting InAppMessaging runtime with Instance ID XXXX
```

XXXX es el identificador de instancia de dispositivo de *In-App Messaging* que usaremos para enviar mensajes de prueba y donde se recibaran a demanda.

Pasos para crear una campaña:

Paso 1: estilo y contenido



Usa las plantillas de Firebase In-App Messaging para incorporar diferentes funciones y estilos a los mensajes.

Estos son los tipos de mensajes que podemos enviar:

Tarjeta **Modal** **Solo imagen** **Banner**

Cada tipo de mensaje dispondrá de unas características:

Característica	Tarjeta	Modal	Solo imagen	Banner
Imagen	●	●	●	●
Acción	●	●	●	●
Color texto/fondo	●	●		●
Título/Cuerpo mensaje	●	●		●
Botón principal	●	●		
Botón secundario	●			

La descripción de cada característica es la siguiente:

- **Imagen:** Proporciona una *URL HTTPS* para la imagen. Puedes usar *Firebase Hosting* o *Firebase Storage* para alojar las imágenes.
- **Acción:** Use enlaces o enlaces profundos para enviar a los usuarios a páginas web o contenido específico en la aplicación. Usa *Firebase Dynamic Links* para crear y administrar enlaces profundos. Las plantillas *Modal* e *Imagen* solo incluyen un botón X, que permite descartar el mensaje.
- **Color texto/fondo:** Personaliza el mensaje con un color específico para el texto o fondo.
- **Título/Cuerpo mensaje:** Capta la atención de los usuarios con un encabezado relevante y una descripción concisa.
- **Botón principal:** La acción predeterminada del botón está configurada para descartar el mensaje. Proporciona una URL para redirigir a los usuarios a la acción deseada.

- **Botón secundario:** Puedes añadir otro botón al mensaje. De forma predeterminada se configurará para descartar el mensaje, pero puedes asociarlo a una determinada acción.

Firebase In-App Messaging proporciona comportamientos preconfigurados y tipos de mensajes con un aspecto predeterminado, pero en algunos casos es posible que desee ampliar los comportamientos y el contenido del mensaje. Podemos agregar acciones y personalizar la apariencia del mensaje.

Mediante acciones en mensajes podemos dirigir a los usuarios a un sitio web o una pantalla específica en la aplicación.

Si la aplicación es la que tiene que procesar la acción debes de implementar el manejador correspondiente. Debe ser capaz de manejar enlaces profundos de *Firebase Dynamic Links*.

Una vez que la aplicación tenga un manejador de enlaces, estará lista para procesar acciones. En la consola de *Firebase* y dependiendo del estilo del mensaje, implementaremos la acción en el mismo mensaje o en uno o varios botones.

El formato de la acción depende del diseño de mensaje que elija. Las tarjetas y los mensajes modales utilizan botones de acción. Puedes personalizar: el texto, color de texto y color de fondo. Las imágenes y los banners, por otro lado, se vuelven interactivos e invocan la acción especificada cuando se pulsa sobre él.

Indicar que también puedes modificar el aspecto de un mensaje In-App Messaging directamente en la aplicación Android con la clase `FirebaseInAppMessagingDisplay`.

Paso 2: Destino

En este paso deberemos especificar el nombre de la campaña y si lo deseamos una descripción. El nombre se usa para los informes de campaña y no forma parte del mensaje visible.

Al igual que con *Firebase Cloud Messaging*, deberemos seleccionar un público objetivo. Seleccionaremos que aplicación se asociará con la campaña. Es posible reducir más los usuarios objetivo, seleccionando especificaciones adicionales con datos obtenidos mediante *Firebase Analytics*. Nos informará del porcentaje de usuarios potenciales.

Paso 3: Programación

Puedes programar una fecha y hora de inicio para la campaña. La campaña puede comenzar cuando se publique la campaña o en un inicio programado.

Podemos establecer la fecha y hora de finalización de la campaña. la campaña puede ejecutarse indefinidamente o tener un final programado.

El mensaje se activa en la aplicación cuando se produce algún evento. Debemos seleccionar estos eventos. Elegiremos entre unos eventos predeterminados o eventos de conversión de *Firebase Analytics*. Estos eventos pueden ser acciones del usuario, eventos del sistema o errores.

Especifique el límite de frecuencia de impresión del mensaje por dispositivo. El límite le permite controlar con qué frecuencia sus usuarios ven su mensaje. De manera predeterminada, una campaña no se muestra después de que el usuario la

haya visto (es decir, impresionado) una vez. O bien, puedes configurar la frecuencia de los mensajes en días.

Paso 4: Eventos de conversión (opcional)

Firebase rastrea la cantidad de impresiones que resultan en un evento de conversión completado. Seleccionar eventos a seguir:

- Eventos de conversión predeterminados.
- Cualquier evento de conversión que haya habilitado.

Después de publicar la campaña podrás ver su historial de los eventos seleccionados.

Paso 5: Opciones adicionales (opcional)

Podemos añadir a la campaña pares clave-valor que se entregarán con el mensaje.

Publicar la campaña

Después de introducir los datos de la campaña, puedes guardarla como borrador para poder editarla posteriormente. O bien, puedes publicarla en la fecha programada.

Puedes editar una campaña después de que se haya publicado. Una vez que detenga una campaña publicada, no se podrá volver publicar. Puedes detener o editar una campaña en ejecución en cualquier momento. También puede duplicar una existente para hacer pequeñas variaciones y evitar crear campañas completamente nuevas.

Vamos a ver como crear una campaña *In-App Messaging* y enviarlo a un dispositivo de prueba:

Ejercicio: *Enviar campañas.*

1. Accede a la consola de *Firebase* al apartado *"In-App Messaging"* que encontrarás en la sección *"Grow"*.
2. Presiona el botón *"Crear campaña"* y rellena los campos de *"Estilo y contenido"* que se indican a continuación.
3. Introduce el título del mensaje: *"Fallas"*
4. Añade la url de la imagen del evento *"Fallas"* de la aplicación *"Eventos"* que deberías tener almacenada en *Firebase Storage* en *"URL de la imagen vertical"* de la sección *"Imágenes"*.

   Para conseguir la url del evento *"Fallas"*, accede a la consola de Firebase, al apartado *Storage*. Pulsa sobre el archivo *"fallas"* y copia el vínculo que aparece debajo del nombre. La url será parecida a la siguiente:

```
https://firebasestorage.googleapis.com/v0/b/eventos.appspot.com/o/fallas?alt=media&token=de5319b1-ef78-4723-9e80-b6caacd7dbb9
```

5. Introduce *"Cerrar"* en el campo *"Texto del botón"* en la sección *"Botón principal"*.
6. Pulsa el botón *"Siguiente"* y rellena los campos de *"Destino"*.
7. Inserta *"Fallas"* en el campo *"Nombre de la campaña"*.
8. Selecciona *"org.example.eventos"* en *"Usuarios aptos"*.
9. Pulsa el botón *"Siguiente"* para rellenar los campos de *"Programación"*.
10. Este apartado lo vamos a dejar sin modificar. Es decir, vamos a enviar el mensaje al instante y se va a activar sólo una vez por dispositivo cuando la aplicación esté en primer plano.
11. Pulsa el botón *"Siguiente"* y rellena los campos de *"Eventos de conversión"*.
12. Tampoco vamos a realizar ningún cambio en este apartado. No vamos a capturar ningún otro evento que no sea el de *"Impresiones"*.
13. Pulsa el botón *"Siguiente"* para rellenar los campos de *"Opciones adicionales"*.
14. No vamos a enviar ningún dato adicional en el mensaje. Lo dejamos igual.
15. Asegurate de cerrar la aplicación Android de *"Eventos"* no esté en primer plano.
16. Pulsa el botón *"Probar en el dispositivo"* que encontrarás en *"Vista previa del dispositivo"*.
17. Nos aparecerá una pantalla donde deberemos introducir el identificador del dispositivo de pruebas que hemos conseguido en el ejercicio anterior.
18. Pulsa el botón *"Prueba"*.
19. Vuelve a ejecutar la aplicación y verifica que se muestra la campaña en la aplicación.

5.4.4. Comportamiento de los mensajes

Firebase In-App Messaging permite crear, configurar y dirigir a los usuarios hacia interacciones con valor. Aprovecha las capacidades de *Google Analytics* y *Predictions* para vincular eventos del mensaje con actividades y opciones de usuario reales y previstas. Podemos adaptar aún más el comportamiento, respondiendo cuando los usuarios interactúan con los mensajes, permitiendo a los usuarios controlar la información que aportan a las respuestas.

Interacción con los mensajes

Usa las acciones de los mensajes para dirigir a los usuarios a un sitio web o contenido dentro de la aplicación. Pero la aplicación Android, por si misma, puede responder a interacciones básicas (clics o rechazos), a impresiones (vistas verificadas de un mensaje) y mostrar errores. Por ejemplo, en un mensaje modal, es posible realizar un seguimiento cual de las dos URL de la tarjeta hizo clic el usuario.

Para realizarlo, debemos implementar en la aplicación Android el detector de eventos del mensaje. Debemos implementar una clase escuchador y luego registrar el detector de eventos con `FirebaseInAppMessaging`.

```
public class inAppClickListener implements
FirebaseInAppMessagingClickListener {
   @Override
   public void messageClicked(InAppMessage inAppMessage, Action action) {
      // Obtiene la url clickada por el usuario
      String url = action.getActionUrl();
      // Obtiene información general de la campaña
      CampaignMetadata metadata = inAppMessage.getCampaignMetadata();
   }
}
```

Registra el escuchador:

```
inAppClickListener inAppListener = new inAppClickListener();
FirebaseInAppMessaging.getInstance().addClickListener(inAppListener);
```

Desencadenar mensajes mediante programación

De forma predeterminada *In-App Messaging* permite activar mensajes en la aplicación para eventos de *Firebase* con *Google Analytics*, sin integración adicional. También puedes activar eventos manualmente mediante programación.

Consigue el identificador de una campaña seleccionando *"Copiar ID de la campaña"* en las opciones del menú de una campaña existente. Modifica la implementación en la aplicación Android para que se dispare la campaña escogida en un punto determinado de la aplicación mediante:

```
FirebaseInAppMessaging.getInstance().triggerEvent("evento");
```

Ejercicio: *Interactuar con mensajes de In-App Messaging.*

1. Crea una nueva campaña con las siguientes características:
   - **Estilo:** Banner superior
   - **Título:** Se suspenden las Fallas
   - **Acción:** URL vínculo dinámico de "Vinculo con sobrecarga"
   - **Nombre de la campaña:** Suspensión de las Fallas
   - **Usuarios que cumplen el requisito:** Aplicación `org.example.eventos`
2. Publica la campaña y desde la lista de campañas
3. Elige la opción "Probar en dispositivo" desde el menú de la campaña recién creada.
4. Añade la siguiente clase al final de la clase `MainActivity`:

```
public class inAppClickListener implements
FirebaseInAppMessagingClickListener {
```

```
    @Override
    public void messageClicked(InAppMessage inAppMessage, Action action) {
        String evento="";
        evento ="ID Campaña: "
                        +inAppMessage.getCampaignMetadata().getCampaignId();
        mostrarDialogo(getApplicationContext(), evento);
    }
}
```

5. Copia y pega el siguiente código al final de método `onCreate` de la clase `MainActivity`:

```
inAppClickListener inAppListener = new inAppClickListener();
FirebaseInAppMessaging.getInstance().addClickListener(inAppListener);
FirebaseInAppMessaging.getInstance()
                              .setAutomaticDataCollectionEnabled(true);
```

6. Ejecuta la aplicación.

Comprueba que se recibe el mensaje y al pulsar sobre él, primero aparece un diálogo con el id de la campaña recibida y posteriormente se muestran los detalles de un evento.

Mediante el escuchador, somos capaces de capturar los eventos que se producen en los mensajes y poder actuar en consecuencia. Si el usuario cancela el mensaje o este no tiene ninguna acción asociada, el escuchador nunca se activará.

Usar metadatos personalizados

En una campaña se pueden especificar datos personalizados, mediante pares clave / valor. La utilización de estos datos personalizados pueden usarse para mostrar un código de una promoción. Los obtendremos mediante un objeto `Bundle` en la clase `FirebaseInAppMessagingClickListener`:

```
public class inAppClickListener implements
FirebaseInAppMessagingClickListener {
    @Override
    public void messageClicked(InAppMessage inAppMessage, Action action) {
        // Conseguir datos personalizado
        Map dataBundle = inAppMessage.getData();
    }
}
```

Desactivar temporalmente los mensajes

De forma predeterminada, Firebase In-App Messaging muestra mensajes cuando se cumple una condición de activación, independientemente del estado actual de una aplicación. Si quieres suprimir la visualización de mensajes, puedes hacerlo con el método `setMessagesSuppressed`.

```
FirebaseInAppMessaging.getInstance().setMessagesSuppressed(true);
```

Con `true`, evitas que se muestren mensajes, y `false`, vuelves a habilitar la su visualización. La supresión de mensajes se desactiva al reiniciar la aplicación. Los

mensajes suprimidos son ignorados. Para que se muestre un mensaje de nuevo, las condiciones de activación deben cumplirse nuevamente.

Habilitar la entrega de mensajes de exclusión

De manera predeterminada, *Firebase In-App Messaging* entrega automáticamente mensajes a todos los usuarios a los que se dirige. Para entregar los mensajes, se usa el identificador de instancia de *Firebase* para identificar la aplicación de cada usuario. Esto significa que los mensajes en la aplicación deben enviar datos del cliente, vinculados a la ID de instancia y a los servidores de *Firebase*. Para dar a los usuarios más control sobre los datos que envían, deshabilita la recopilación automática de datos.

Para hacer eso, debes deshabilitar la inicialización automática e inicializar el servicio manualmente:

1. Deshabilita la inicialización automática añadiendo el siguiente `meta-data` en el manifiesto:

```
<meta-data
   android:name="firebase_inapp_messaging_auto_data_collection_enabled"
   android:value="false" />
```

2. Inicializa la instancia de *Firebase In-*App Messaging de forma automática, mediante:

```
FirebaseInAppMessaging.getInstance()
                                .setAutomaticDataCollectionEnabled(true);
```

Cuando se establece una preferencia de recopilación de datos manualmente, el valor persiste a través del reinicio de la aplicación, anulando el valor en Manifest. Si desea deshabilitar la inicialización nuevamente, por ejemplo, si un usuario opta por dejar de recopilar más tarde, pase false al método `setAutomaticDataCollectionEnabled`.

5.5. Firebase Stability

Vamos a ver una introducción al apartado *Firebase Stability* en la consola de Firebase. Está compuesta por los siguientes apartados: *Crashlytics*, *Performance* y *Test Lab*. Este conjunto de herramientas está pensado para otorgar estabilidad a la aplicación. Obteniendo informes de errores y rendimiento directamente desde los usuarios que utilizan la aplicación. Así como poder realizar un testeo de nuestra aplicación en diferentes dispositivos y configuraciones.

5.5.1. Crashlytics

Firebase Crashlytics es una herramienta de informes de fallos. Proporciona información sobre los problemas de una aplicación. Funciona en iOS y Android.

Es una herramienta ligera que informa de fallos en tiempo real, ayuda a realizar un seguimiento, priorizar y corregir problemas. Agrupa los fallos de forma inteligente

y destaca las circunstancias en las que se produjeron, esto permite encontrar soluciones más rápidamente.

Descubre si un fallo afecta a muchos usuarios. Cuando la gravedad de un problema aumenta puedes recibir alertas. Localiza qué líneas de código provocan los errores.

Las principales funciones de Crashlytics son:

- **Informes de fallos con datos seleccionados**: Crashlytics evalua y sintetiza una gran cantidad de fallos en una lista de problemas manejable. Además, proporciona información del contexto en el que se produce el error. Destaca la gravedad y prevalencia de los fallos para determinar la causa que lo provoca más rápidamente.

- **Información mejorada del entorno:** Permite filtrar informes por: sistema operativo, configuración de hardware, versión, ... Puedes descubrir si la aplicación falla más en dispositivos de un fabricante o en una orientación de pantalla.

- **Alertas en tiempo real:** Recibe alertas en tiempo real de problemas nuevos, recurrentes y crecientes. Son los que deberían requerir una atención más inmediata.

Para usar *Crashlytics* en una aplicación Android debemos seguir los siguientes pasos:

1. Conecta la aplicación: Comienza por agregar Firebase a la aplicación.

2. Integra el SDK: Agrega las dependencias de *Gradle* y código para inicializar los informes de Crashlytics.

3. Verifica los informes en la consola de Firebase: Utiliza la consola de Firebase para identificar los problemas que afectan a los usuarios y descubrir sus causas.

5.5.1.1. Inicializar Crashlitycs en Android.

Veamos como inicializar *Crashlitycs* en la consola de *Firebase*:

Ejercicio: *Inicializar Firebase Crashlytics en la consola Firebase.*

1. Accede a la consola de *Firebase* con la cuenta con la que has creado el proyecto *"Eventos"*.
2. Selecciona el proyecto *"Eventos"*.
3. En el apartado *"Stability"* selecciona *"Crashlytics"*.
4. Pulsa en el botón *"Configurar Crashlytics"*.

Crashlitys es el sustituto de “Crash Reporting” y se basa en “Fabric”.

5. Tenemos que usar un asistente para inicializar *Crashlitycs*. En el primer apartado *“¿Eres un usuario de Fabric que quiere migrar una app de Crashlytics?”* elegimos *“No, quiero configurar una app de Firebase nueva”.*
6. En el segundo paso del asistente *“Instala el SDK”*, nos pide que instalemos el SDK en la aplicación Android. Pulsa en el botón *“Ir a los documentos de Crashlytics”* para leer la documentación de cómo hacerlo. Lo veremos en el siguiente ejercicio.
7. En el último apartado nos indica que para empezar a utilizar *Crashlytics* solo tenemos que compilar y ejecutar la aplicación y esperar a que se produzca algún error.

   Veamos como configurar *Crashlytics* en una aplicación Android.

Ejercicio: *Configurar Firebase Crashlytics en una app Android.*

1. Tenemos que tener configurado Firebase en la aplicación Android. Si realizas este ejercicio en el proyecto “Eventos”, ya lo tenemos hecho en ejercicios anteriores.
2. En el fichero *build.gradle (Module)* del proyecto, agrega dentro de `repositories` en `buildscripts`:

```
maven {
   url 'https://maven.fabric.io/public'
}
```

3. En el mismo fichero *build.gradle (Module)* añade en `dependencies` en `buildscripts`:

```
classpath 'io.fabric.tools:gradle:1.+'
```

4. Seguimos en el mismo archivo *build.gradle (Module)*. Inserta dentro de `repositories` en `allprojects`:

```
maven {
url 'https://s3.amazonaws.com/fabric-artifacts-private/internal-snapshots'
}
maven {
   url 'https://maven.fabric.io/public'
}
```

5. Inserta debajo de la línea `apply plugin: 'com.android.application'` en *build.graddle (Project)* el siguiente `plugin`:

```
apply plugin: 'io.fabric'
```

6. Añade las siguientes dependencias en el fichero anterior:

```
implementation
        ('com.crashlytics.sdk.android:crashlytics:2.7.0-SNAPSHOT@aar') {
    transitive = true;
}
```

7. Agrega la cadena `useFirebaseAppId` a strings.xml:

```
<string name="com.crashlytics.useFirebaseAppId">true</string>
```

8. Cambia la vista del proyecto de *Android* a *"Project Files"* y crea un archivo *app/fabric.properties* con la línea siguiente:

```
USE_FIREBASE_APP_ID=true
```

9. Importa en la clase `MainActivity`, las clases `Crashlytics` y `Fabric`:

```
import com.crashlytics.android.Crashlytics;
import io.fabric.sdk.android.Fabric;
```

10. Inicializa Crashlytics añadiendo el siguiente código al final del método `onCreate()` de la clase `MainActivity`:

```
Fabric.with(this, new Crashlytics());
```

11. Inserta la siguiente opción de menú en el fichero `main_menu.xml` que encontrarás en la carpeta `menu` de los recursos:

```
<item
    android:id="@+id/action_error"
    android:orderInCategory="101"
    android:title="Error"
    app:showAsAction="never" />
```

12. Añade debajo de `int id=item.getItemId();` del método `onOptionsItemSelected` en la clase `MainActivity` el siguiente código:

```
if (id == R.id.action_error) {
    Crashlytics.getInstance().crash();
    return true;
}
```

No es necesario esperar un fallo para saber si Crashlytics está funcionando. Puedes utilizar el método: `Crashlytics.getInstance().crash()` para forzar un fallo. En nuestro caso lo provocaremos cuando seleccionemos el menú "Error" en la pantalla principal.

13. Accede a https://console.firebase.google.com/subscriptions/overview para activar las alertas de Crashlytics.

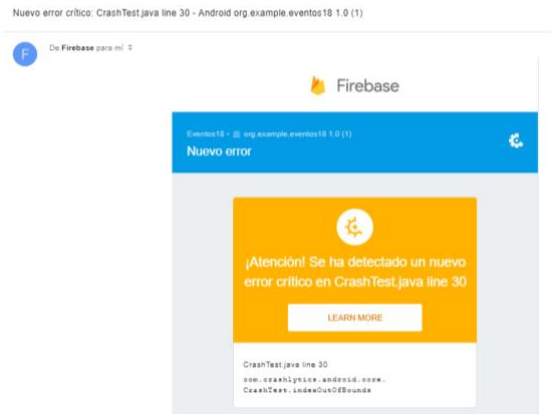
14. Selecciona el proyecto "Eventos".

15. Activa todas las opciones de notificación en el apartado *Crashlytics*.

Hemos activado que se nos informe de todos los errores mediante la consola de Firebase y correo electrónico.

16. Ejecuta la aplicación y provoca un fallo.

Elige la opción “Error” en la pantalla principal para forzar un error. Comprueba que recibes casi inmediatamente un email informando del error en la cuenta de desarrollo. El email recibido será similar al de la imagen.

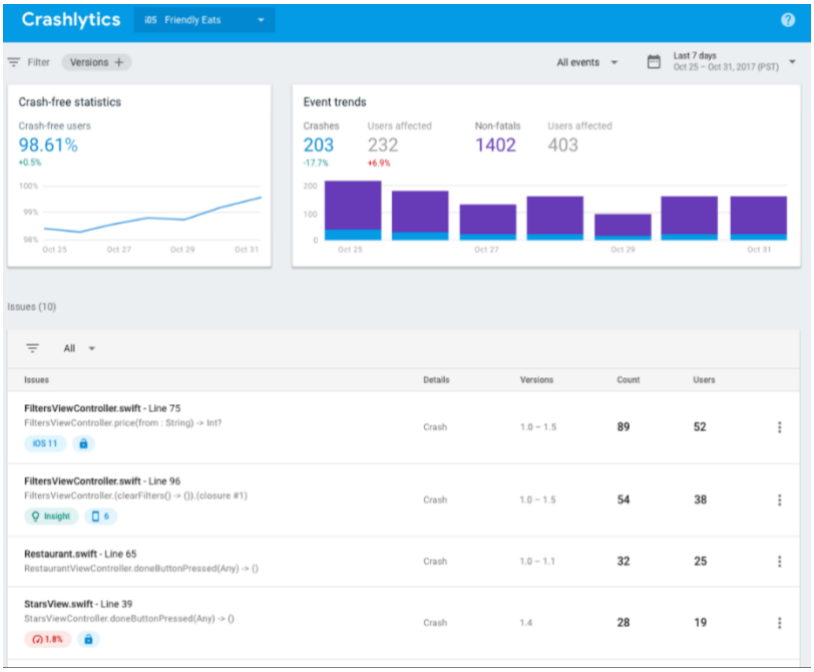


Si pulsamos en “LEARN MORE”, accederemos a la consola de Firebase, directamente a la información del error. Donde podremos ver exactamente, además de otra mucha información, que línea de que actividad es la que ha producido el error.

Desde la consola de Firebase puedes mantener un control de fallos solucionados. Cuando accedes a los detalles de un error en la consola, tienes el botón “SILENCIAR”. Si lo pulsas evitarás que se reabra la incidencia, incluso en futuras versiones.

Con Firebase Crashlytics puedes supervisar los informes de fallos desde Firebase console:

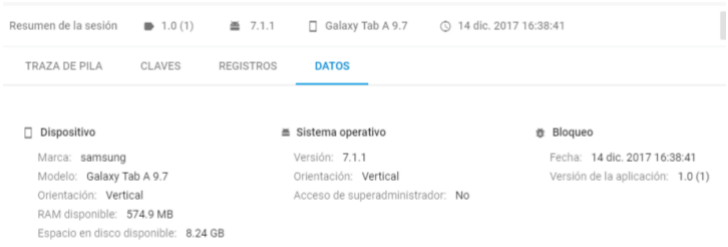
Consulta los informes de errores desde la consola de Firebase. Abre la consola de *Firebase* y selecciona el proyecto de la aplicación *“Eventos”*. Selecciona *Crashlytics* en la barra de navegación del lado izquierdo.



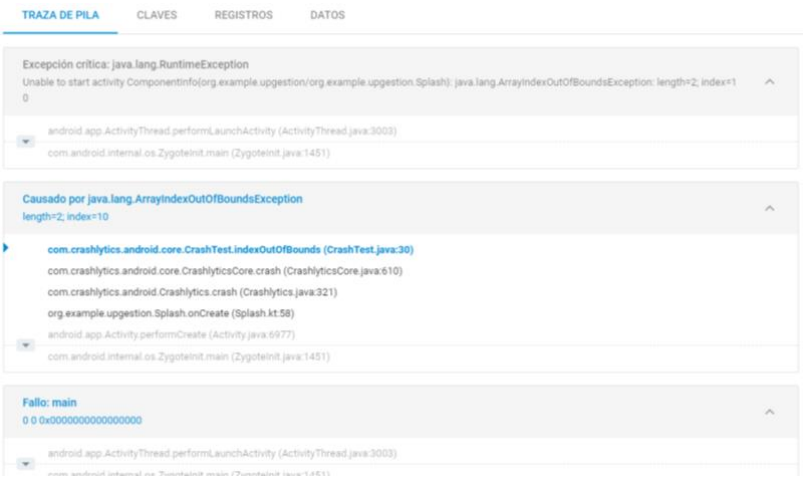
Tenemos tres paneles con información. Arriba a la izquierda tenemos el gráfico diario de "Estadísticas de la ausencia de fallos" que informa del porcentaje de usuarios que no han tenido ningún fallo. Arriba a la derecha, tenemos el gráfico diario de "Tendencias del evento" que informa de los errores. Distingue entre errores de bloqueo y de no bloqueo, además de a cuantos usuarios ha afectado cada tipo de error. En la parte de abajo tenemos una tabla con los distintos errores.

Podemos ver diferentes tipos de gráficos, cómo los que nos muestran el número de errores por: versión de la aplicación, dispositivo, versión del sistema operativo y estado de los dispositivos (si se produce en primer o segundo plano).

Finalmente podemos consultar las sesiones de cada error. Será donde obtendremos información de un mismo error, que se ha producido en diferentes usuarios, de forma unitaria. Nos informará desde las características del dispositivo, como puedes ver en la siguiente imagen.



Además de la traza del error, así obtendremos la misma información que obtenemos del *Logcat* de *Android Studio* pero de cada usuario:



Recuerda quitar todas las lineas `Crashlytics.getInstance().crash();` antes de publicar la aplicación. Esta instrucción sólo sirve para provocar errores y realizar pruebas.

Crashlytics comienza a recopilar informes de fallos automáticamente, sólo tienen que agregar el SDK. Podemos personalizar la configuración con informes opcionales, registros y claves, también podemos hacer seguimiento de errores recuperables.

5.5.1.2. Habilitar los informes de inclusión voluntaria

Por defecto, Firebase Crashlytics, recopila automáticamente informes de fallos de todos los usuarios de la aplicación. Para dar a los usuarios más control sobre los datos que envían, podemos habilitar los informes de inclusión voluntaria. De esta forma sólo se recopilarán datos de los usuarios que acepten participar.

Para activar los informes de inclusión automática en una aplicación Android, tienes que deshabilitar la recopilación automática de fallos e inicializar Crashlytics solo para los usuarios que acepten participar.

Para desactivar la recolección automática, incluye dentro de *`application`* en el manifiesto la siguiente etiqueta

```
<meta-data android:name="firebase_crashlytics_collection_enabled"
                                                android:value="false" />
```

Habilita la recopilación de datos mediante la siguiente instrucción en alguna actividad de la aplicación:

```
Fabric.with(this, new Crashlytics());
```

En el ejercicio anterior ya hemos inicializado la recopilación de datos incluyendo esta instrucción en el método `onCreate` de la clase `MainActivity`. Si quisieras

activar la inclusión voluntaria, solamente debería ejecutar la instrucción los usuarios que deseen participar.

Durante la ejecución de la aplicación no se puede detener la recopilación de datos para los informes de Crashlytics una vez que se haya aceptado participar. Para inhabilitar los informes después de inicializar *Crashlytics*, los usuarios deben reiniciar la aplicación y revocar la autorización a participar.

Práctica: *Activar informes de inclusión voluntaria.*

Permite a los usuarios elegir si quieren participar en el envío de informe de fallos. Para ello, preguntales mediante un cuadro de diálogo al iniciar por primera vez la aplicación lo siguiente:

"Con el fin de mejorar la aplicación, te pedimos que participes en el envío automático de errores a nuestros servidores. ¿Estás de acuerdo?"

Si el usuario acepta, guardaremos su respuesta en las preferencias y activaremos la captura de fallos. Si no acepta, también guardaremos su respuesta y no le volveremos a preguntar.

5.5.1.3. Agregar registros personalizados

Para tener más información del contexto previo en el que se produce un error, podemos agregar registros de Crashlytics personalizados a la aplicación. Crashlytics asocia los registros con sus datos de bloqueo y los hace visibles en la consola de Firebase.

En Android, utiliza el método `Crashlytics.log` para ayudar a identificar problemas. `Crashlytics.log` escribe registros en un informe de fallos, además opcionalmente puede mostrar la información en el `Log`:

- Para mostar información en *Firebase Crashlytics* y el *Log* utiliza:

```
Crashlytics.log(int priority, String tag, String msg);
```

- Para mostrar información solamente en *Firebase Crashlytics*:

```
Crashlytics.log(String msg);
```

Para evitar ralentizar de la aplicación, Crashlytics limita los registros a 64kB. Elimina entradas de registro antiguas si los registros de una sesión sobrepasan ese límite.

5.5.1.4. Añadir claves personalizadas

Las claves personalizadas ayudan a obtener el estado específico de la aplicación hasta que ocurre un error. Podemos asociar pares de clave/valor arbitrarios a los informes y verlos en la consola de Firebase.

Hay cinco métodos para establecer claves. Cada uno maneja un tipo de datos diferente:

```
Crashlytics.setString ( key , value );
Crashlytics.setBool ( String key , boolean value );
Crashlytics.setDouble ( String key , double value );
Crashlytics.setFloat ( String key , float value );
Crashlytics.setInt ( String key , int value );
```

Veamos algún ejemplo para actualizar el valor de una clave:

```
Crashlytics.setInt("nivel_actual", 3);
Crashlytics.setString("ultima_accion_UI","menu_suscripciones");
```

Crashlytics admite un máximo de 64 pares clave/valor. Una vez que alcanza este umbral, los valores adicionales no se guardan. Cada par clave / valor puede tener un tamaño de hasta 1 kB.

5.5.1.5. Establecer ID de usuario.

Para diagnosticar un problema, a menudo es útil saber cuál de los usuarios experimentó un bloqueo determinado. *Crashlytics* incluye una forma de identificar anónimamente a los usuarios en los informes de fallos.

Para agregar un ID de usuario, asigna a cada usuario un identificador único. Puede ser un número, token o valor hash:

```
    void Crashlytics.setUserIdentifier(String identifier);
```

Si necesita borrar un identificador de usuario después de establecerlo, restablezca el valor a una cadena en blanco.

5.5.1.6. Registrar excepciones no fatales

Además de informar automáticamente de errores, *Crashlytics* permite registrar excepciones no fatales. En Android, significa que puede registrar excepciones detectadas en los bloques `catch` de la aplicación:

Todas las excepciones registradas aparecen como problemas no fatales en la consola de Firebase. El resumen del problema contiene toda la información de estado que normalmente recibe de los bloqueos, como la versión de Android y el hardware del dispositivo.

Crashlytics procesa las excepciones en un hilo dedicado, por lo que el impacto en el rendimiento es mínimo. Para reducir el tráfico de red, *Crashlytics* combina las excepciones registradas y las envía la próxima vez que se inicia la aplicación.

Crashlytics solo almacena las 8 excepciones más recientes de una sesión. Si su aplicación arroja más de 8 excepciones en una sesión, se pierden las excepciones anteriores.

5.5.1.7. Administrar datos de Crash Insights

Crash Insights te ayuda a resolver problemas al comparar seguimientos con los de otras aplicaciones de Firebase y permitiéndote saber si tu problema es parte de una tendencia más amplia. Para muchos problemas, *Crash Insights* incluso proporciona recursos para ayudar a depurar el bloqueo.

Crash Insights utiliza datos acumulados de bloqueos para identificar tendencias comunes de estabilidad. Si prefieres no compartir los datos de la aplicación, puedes cancelar la suscripción a *Crash Insights* en el menú de *Crash Insights* en la parte superior de la lista de problemas de *Crashlytics* en la consola de *Firebase*.

5.5.2. Performance

Obtén estadísticas útiles sobre el rendimiento y las latencias que experimentan los usuarios de una aplicación. *Firebase Performance Monitoring* es un servicio que ayuda a obtener estadísticas sobre las características de rendimiento de aplicaciones en iOS y Android. Usa *Performance Monitoring* para recopilar datos de rendimiento y, luego, revisa y analiza los datos en la consola de Firebase. Te ayuda a comprender dónde y cuándo se puede mejorar el rendimiento de una aplicación.

Mide de forma automática el tiempo de inicio de la aplicación, las solicitudes de red HTTP/S y mucho más. Cuando integras *Performance Monitoring* en una aplicación, no necesitas escribir ningún código para que comience a supervisar varios aspectos críticos del rendimiento, como el tiempo de inicio, la actividad en primer plano, la actividad en segundo plano y las solicitudes de red HTTP/S.

Obtén estadísticas sobre las situaciones en las que se puede mejorar el rendimiento. La optimización del rendimiento puede ser un desafío cuando no sabes exactamente por qué no se cumplen las expectativas del usuario. Por esto, *Performance Monitoring* permite ver métricas de rendimiento desglosadas por país, dispositivo, versión de la aplicación y sistema operativo.

Puedes personalizar *Performance Monitoring* para una aplicación. Puedes crear seguimientos para registrar el rendimiento en situaciones específicas, como cuando cargas una pantalla nueva. Además, puedes crear contadores para llevar un recuento de los eventos que defines.

Un seguimiento es un informe de rendimiento capturado en un período de tiempo. Cuando se instala, *Performance Monitoring* proporciona seguimientos del inicio de forma automática, que miden el tiempo desde que el usuario abre la aplicación hasta que está lista para responder. Además, proporciona seguimientos

en primer y segundo plano para mostrar estadísticas de rendimiento cuando está activa o inactiva.

Podemos configurar seguimientos personalizados. Un seguimiento personalizado es un informe de datos de rendimiento asociados con una parte del código de la aplicación. Define el inicio y el final de un seguimiento personalizado. Puedes configurar un seguimiento personalizado que contabilize el rendimiento de un determinado evento. Por ejemplo, puedes crear un contador de la cantidad de veces que la interfaz de usuario deja de responder.

Una solicitud de red HTTP/S es un informe que captura el tiempo desde que la aplicación envía una solicitud a un extremo del servicio hasta que se completa la respuesta de ese extremo. Captura varias métricas sobre cualquier extremo al que se envíe una solicitud:

- **Tiempo de respuesta**: tiempo transcurrido desde que se envía la solicitud hasta que se recibe la respuesta por completo.
- **Tamaño de la carga útil**: tamaño en bytes de la carga útil de red que descargó o subió la aplicación.
- **Tasa de éxito:** porcentaje de respuestas correctas en comparación con el total de respuestas (para medir errores de la red o del sistema).

Tanto para los seguimientos como para las solicitudes de red HTTP/S, puedes ver los datos de supervisión del rendimiento ordenados según las siguientes categorías:

- **Seguimientos:** versión de la aplicación, país, dispositivo, SO, radioteléfono y proveedor.
- **Solicitudes de red HTTP/S:** versión de la aplicación, país, dispositivo, SO, radioteléfono, proveedor y tipo MIME.

Performance Monitoring nunca almacena información de identificación personal de forma permanente (como nombres, direcciones de correo electrónico o números de teléfono). Para la supervisión de solicitudes de red HTTP/S, *Performance Monitoring* usa URL (sin incluir parámetros) para crear patrones de URL generales y anónimos, son los que se conservan y se muestran en la consola de Firebase.

5.5.2.1. Performance Monitoring en Android

No tenemos que activar nada en la consola de *Firebase*, simplemente instala del SDK de *Performance Monitoring* en la aplicación Android. Los datos aparecerán en la consola hasta 12 horas después de haberse capturado en el dispositivo Android.

Ejercicio: *Performance Monitoring en Android.*

1. Abre el archivo `build.gradle (Project)` de proyecto y agrega lo siguiente:

   En la sección `buildscript` -> `repositories`:

```
jcenter()
```

   En la sección `buildscript` -> `dependencies`:

```
classpath 'com.google.firebase:firebase-plugins:1.1.1'
```

   Seguramente `jcenter()` ya lo tengas incluido.

2. Abre el archivo `build.gradle (module)` de la aplicación y agrega debajo de `apply plugin: 'com.android.application'`:

```
apply plugin: 'com.google.firebase.firebase-perf'
```

   Agrega en la sección `dependencies`:

```
implementation 'com.google.firebase:firebase-perf:17.0.0'
```

3. Vuelve a compilar la aplicación y ejecútala.

Ahora, se supervisarán los seguimientos automáticos y las solicitudes de red HTTP/S. Los datos empezaran a mostrarse hasta 12 horas después de haberse recopliado en la consola de Firebase.

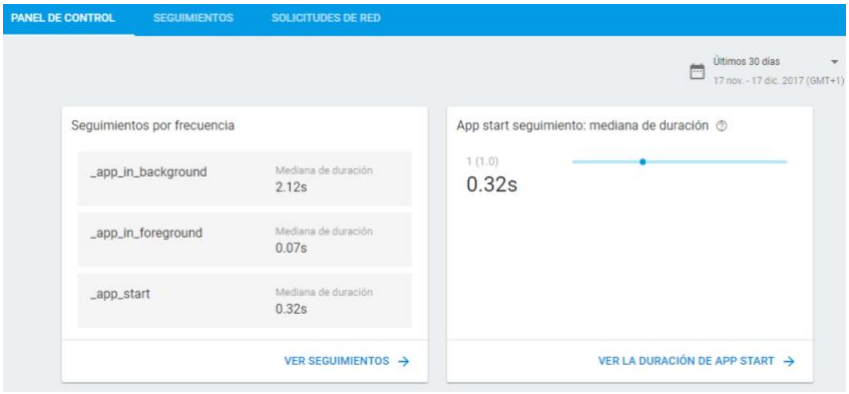
Accede a la consola de *Firebase* y pulsa sobre el menú *"Performance"* del apartado *"Stability"*. Podremos consultar los seguimientos automáticos. Un seguimiento es un informe de datos de rendimiento que se capturan en un período de tiempo. *Performance Monitoring* proporciona automáticamente los siguientes tipos de seguimientos:

- **Seguimientos de inicio de aplicación (_app_start):** miden el tiempo desde que el usuario abre la aplicación hasta que está lista para responder. Se inicia cuando el `ContentProvider` de `FirebasePerfProvider` completa su método `onCreate` y se detiene cuando se llama al método `onResume()` de la actividad.

- **Seguimientos de la aplicación en segundo plano (_app_in_background):** miden el tiempo que la aplicación se ejecuta en segundo plano. Se inicia cuando se llama al método `onStop()` de la última actividad en abandonar el primer plano y se detiene cuando se llama al método `onResume()` de la primera actividad en llegar al primer plano.

- **Seguimientos de la aplicación en primer plano (_app_in_foregorund):** miden el tiempo que la aplicación se ejecuta en primer plano y está disponible para el usuario. Se inicia cuando se llama al método `onResume()` de la primera actividad en llegar al primer plano y se detiene cuando se llama al método `onStop()` de la última actividad en abandonar el primer plano.

Puedes consultar las estadísticas de los seguimientos automáticos en la consola de Firebase.



5.5.2.2. Inhabilita Firebase Performance Monitoring

Es recomendable que los usuarios acepten o rechacen la opción de usar *Firebase Performance Monitoring*. Configura la aplicación, de manera que se pueda habilitar o deshabilitar.

Esta opción también puede ser útil durante la programación y las pruebas de la aplicación. Puedes inhabilitar *Performance Monitoring* cuando compilas la aplicación con la opción de volver a habilitarlo en tiempo de ejecución, o compilar con *Performance Monitoring* habilitado y, luego, tener la opción de inhabilitarlo en tiempo de ejecución con *Firebase Remote Config*. Además, puedes desactivar *Performance Monitoring* por completo, sin la opción de habilitarlo en tiempo de ejecución.

Inhabilitar Performance Monitoring durante la compilación

Una situación en la que inhabilitar *Performance Monitoring* durante el proceso de compilación podría ser útil es para evitar recoger datos de rendimiento de una versión previa al lanzamiento de la aplicación durante la programación y las pruebas.

Para inhabilitar los seguimientos automáticos (pero no los personalizados) y la supervisión de solicitudes de red HTTP/S durante la compilación, agrega la siguiente propiedad al archivo *gradle* de la aplicación dentro del apartado `android`:

```
android.applicationVariants.all {
   FirebasePerformance {
      instrumentationEnabled false
   }
}
```

Cambiar esta propiedad a `true` vuelve a habilitar los seguimientos automáticos y la supervisión de solicitudes de red HTTP/S.

Además, puedes inhabilitar *Performance Monitoring* durante la compilación para posteriormente habilitarlo en tiempo de ejecución. Agregar el siguiente elemento `<meta-data>` al manifiesto:

```
<application>
   <meta-data android:name="firebase_performance_collection_enabled"
                                                android:value="false" />
</application>
```

Para desactivar *Performance Monitoring* por completo sin la opción de habilitarla en tiempo de ejecución, agrega el siguiente elemento `<meta-data>`:

```
<application>
<meta-data android:name="firebase_performance_collection_deactivated"
                                                android:value="true" />
</application>
```

Inhabilitar Performance Monitoring durante tiempo de ejecución

Podemos deshabilitar *Performance Monitoring* en tiempo de ejecución ejecutando el método:

```
FirebasePerformance.getInstance().setPerformanceCollectionEnabled(false);
```

La volveremos a activar llamando al mismo método, pero con valor `true`.

Remote Config te permite hacer cambios en el comportamiento y el aspecto de la aplicación, lo que proporciona una manera ideal de inhabilitar *Performance Monitoring* en instancias de forma remota.

Práctica: *Configurar Performance Monitoring con Remote Config.*

Queremos controlar *Performance Monitoring* de forma remota. Así activaremos o desactivaremos *Performance Monitoring* para todos los usuarios de nuestra aplicación en el momento que queramos. No dependeremos de que tenga una determinada versión de la aplicación para que capte datos o no. Para ello crea un parámerto en *Remote Config* en la consola de *Firebase* llamado `PerformanceMonitoring`. Si tiene el valor `0` no se capturarán datos y si tiene el valor `1` si que se capturarán.

Realiza los cambios necesarios en la aplicación Android. Recuerda utilizar el método:

```
FirebasePerformance.getInstance().setPerformanceCollectionEnabled(valor);
```

5.5.2.3. Seguimientos personalizados

Un seguimiento personalizado es un informe de rendimiento que asociamos a una parte del código de la aplicación. Es posible tener varios seguimientos personalizados en una aplicación y pueden estar ejecutándose al mismo tiempo. Cada seguimiento puede tener uno o más contadores asociados a eventos. Esos contadores están asociados con los seguimientos que los crean.

Vamos a realizar un seguimiento personalizado a la actividad `EventoDetalles`:

Ejercicio: *Crear seguimientos personalizados.*

1. Importa las siguientes clases en la clase `EventoDetalles`:

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

2. Declara la siguiente variable en la clase `EventoDetalles`:

```
Trace mTrace;
```

3. Inserta el siguiente código al final del método `onCreate` de la clase `EventoDetalles`:

```
mTrace =
      FirebasePerformance.getInstance().newTrace("trace_EventoDetalles");
mTrace.start();
```

   Utilizaremos un objeto `Trace` para realizar el seguimiento. Lo inicializamos con `FirebasePerformance.getInstance().newTrace("nombre_traza")`, donde especificaremos el nombre que deseamos dar al seguimiento (en el ejemplo `nombre_traza`). Iniciaremos el seguimiento con el método `start()`.

4. Añade el siguiente código al final de la clase `EventoDetalles`:

```
@Override
protected void onResume(){
   super.onResume();
   mTrace.start();
}
@Override
protected void onStop(){
   super.onStop();
   mTrace.stop();
}
```

   Iniciamos y paramos el seguimiento con los métodos `start()` y `stop()` del objeto `Trace`.

5. Ejecuta la aplicación.

Entra varias veces en la actividad `EventoDetalles` manteniéndola abierta en primer plano con diferentes duraciones. Observa las estadísticas capturadas en la consola de *Firebase*.

Podemos asociar seguimientos a métodos específicos. Utiliza la anotación `@AddTrace` a los métodos que desees monitorizar y proporciona una cadena que identifique la traza. La traza comienza al inicio del método y se para cuando se

completa su ejecución. Las trazas creadas con este método no tienen contadores asociados.

```
@Override
@AddTrace(name = "onCreateTrace", enabled = true)
protected void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   ...
   ...
}
```

Por defecto, obtendremos la duración del seguimiento, es decir, el tiempo que transcurre entre los métodos `start()` y `stop()`. Pero podemos asociar métricas personalizadas a un seguimiento. Puede ser interesante contar eventos que son relevantes para evaluar el rendimiento de la aplicación. Por ejemplo, contar el número de veces que usamos el disco local, que llamamos al GPS, cuantas veces realizamos una llamada de red, …

Debes proporcionar un nombre para cada uno de estos eventos. Se trata de un contador incremental. Debes colocar los contadores entre los métodos `start()` y `stop()` de la traza. Estos contadores se asociarán a la traza y podrán ser consultadas en la consola de Firebase.

Estos son algunos ejemplos de contadores:

```
ClipData.Item item = cache.fetch("item");
if (item != null) {
    mTrace.incrementCounter("item_cache_exito");
} else {
   mTrace.incrementCounter("item_cache_error");
}

mTrace.incrementCounter("llamada_disco");
mTrace.incrementCounter("frame_perdido");
```

5.5.3. Test Lab

Prueba una aplicación en dispositivos alojados en los servidores de Google. *Firebase Test Lab* ofrece una infraestructura basada en la nube para probar aplicaciones Android. Con una sola operación, puedes comenzar a probar la aplicación en una amplia variedad de dispositivos y configuraciones. Los resultados de las pruebas (que incluyen registros, videos y capturas de pantalla) aparecen en la consola de *Firebase*. Incluso sin escribir código de prueba, *Test Lab* puede evaluarla automáticamente en busca de bloqueos.

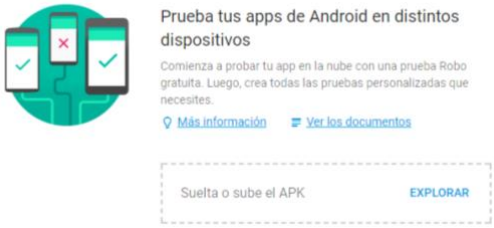
Usa *Test Lab* para evaluar una aplicación en dispositivos reales instalados en un centro de datos de Google. Te ayuda a encontrar problemas que solo ocurren en configuraciones de dispositivos específicas (por ejemplo, un Pixel 2 con un nivel de API de Android en particular con una configuración regional específica).

Evalúa una aplicación sin tener que escribir pruebas antes. Gracias a la prueba *Robo*, puedes identificar problemas en la aplicación sin tener que escribir pruebas. La prueba *Robo* analiza la estructura de la interfaz de usuario y la explora mediante una simulación automática del uso de las actividades por un usuario. Si escribiste pruebas de instrumentación, *Test Lab* también puede ejecutar esas pruebas.

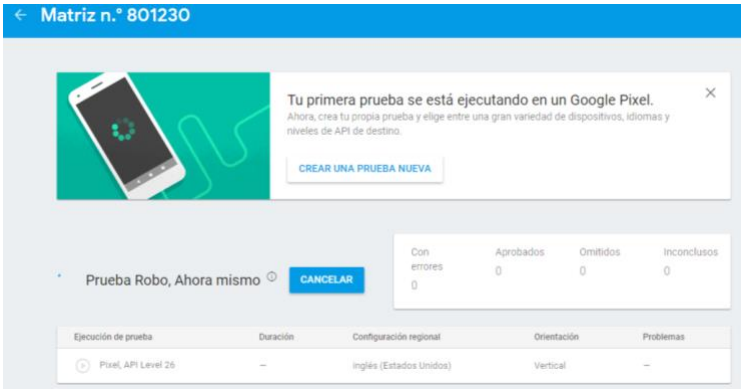
Los dispositivos que se usan para las pruebas son dispositivos reales en los que se cargan niveles de API de Android actualizados o configuraciones regionales según tus especificaciones. Esto permite que hagas pruebas de uso en un conjunto de dispositivos y configuraciones reales.

Ejercicio: *Prueba Robo en Firebase Test Lab.*

1. Selecciona la opción *"Test Lab"* en el apartado *"Stability"* de la consola de Firebase del proyecto *"Eventos".*
2. Genera el apk de la aplicación *"Eventos"* y súbelo a *"Test Lab".*



3. Genera el fichero APK de la aplicación seleccionando la opción de menú *Build > Build Bundle(s) / APK(s) > Build APK(s)*
4. Localiza el fichero APK generado en la carpeta *x:\carpeta_proyecto\app\build\outputs\apk\debug*
5. Se va a generar una primera prueba *Robo* automáticamente. Puede tardar varios minutos en terminar, puedes cerrar la ventana y consultar los resultados más tarde.



Cuando se termine de procesar la prueba, recibirás un email indicando que ya están disponibles los resultados.

6. Vuelve a acceder a la consola de *Firebase*, si la has abandonado. Ves al apartado *Test Lab* y selecciona la prueba que has realizado.

Test Lab te permite ejecutar pruebas de instrumentación *Espresso* y *UI Automator 2.0* escritas para evaluar una aplicación en la consola de Firebase, Android Studio o la interfaz de línea de comandos de *gcloud*. También puedes usar la prueba *Robo* para evaluar una aplicación automáticamente en *Firebase* o en la línea de comando de *gcloud*.

La prueba *Robo* guarda registros, crea un "mapa de actividad" que muestra un conjunto de capturas de pantalla relacionadas con comentarios y genera un video a partir de una secuencia de capturas de pantalla para mostrarte las operaciones que realizó el usuario simulado.

Si vas a ejecutar pruebas de instrumentación, escribe tu propia prueba específica para la aplicación. Cuando desarrolles pruebas de instrumentación, no olvides agregar la biblioteca de capturas de pantalla de Test Lab a tu proyecto de prueba, de esta manera podrás interpretar los resultados más fácilmente.

Elige un entorno y una matriz de pruebas. Elige un ambiente de prueba (*Firebase console*, *Android Studio* o la interfaz de línea de comando *gcloud*) y define una matriz de pruebas. Esto implica seleccionar un conjunto de dispositivos, niveles de API, configuraciones regionales y orientaciones de pantalla.

Ejecuta tus pruebas y revisa los resultados. Según las dimensiones de la matriz de pruebas, es posible que *Test Lab* demore varios minutos en completar su ejecución. Cuando termine el proceso, podrás revisar los resultados de las pruebas en la consola de Firebase.

Prueba Robo

La prueba *Robo* captura los archivos de registro, guarda una serie de capturas de pantalla con anotaciones y crea un video a partir de las capturas de pantalla para mostrarte las operaciones que realizó el usuario simulado. Estos registros, capturas

de pantalla y videos te pueden ayudar a determinar la causa de bloqueos y a encontrar errores en la IU.

Puedes configurar la prueba *Robo* de varias formas:

- **Profundidad máxima.** Para configurar la profundidad máxima con la que explora la prueba *Robo*. Indica a la prueba con cuánta rigurosidad debe explorar una rama en particular de la IU antes de volver a la raíz de la IU (la pantalla principal) para explorar otra rama. El valor predeterminado para la profundidad máxima es 50 y cualquier valor inferior a 2 impide que la prueba explore la aplicación más allá de la pantalla principal.

- **Tiempo de espera.** Según la complejidad de la IU, la prueba *Robo* podría tardar cinco minutos o más en completar un conjunto meticuloso de interacciones de IU. Te recomendamos asignar al tiempo de espera de la prueba al menos 120 segundos (2 minutos) para la mayoría de las aplicaciones, y 300 segundos (5 minutos) para las aplicaciones de complejidad moderada. El valor predeterminado del tiempo de espera es de 300 segundos (5 minutos) para las pruebas que se ejecutan desde Android Studio y Google Developer Console, y de 1.500 segundos (25 minutos) para las pruebas que se ejecutan desde la línea de comandos gcloud.

Si tu aplicación tiene una IU con una profundidad considerable y varias ramas de IU que pueden explorarse, es recomendable que uses algunas de las siguientes opciones para garantizar que la prueba *Robo* explore la aplicación de forma completa:

- Establece un valor alto para el tiempo de espera, de manera que la prueba *Robo* pueda explorar varias ramas de la IU.

- Establece un valor bajo para la profundidad máxima, de manera que la prueba *Robo* explore hasta cierto punto cada rama de la IU.

Puedes usar la prueba *Robo* en Google Play Console cuando subes y publicas el archivo APK con el canal Alpha o el Beta. La prueba *Robo* se ejecuta en un conjunto común de dispositivos físicos desde diferentes ubicaciones geográficas, por lo que la prueba se realiza en varios factores de forma y configuraciones de hardware.

La prueba *Robo* permite acceder a cuentas de prueba y también permite ingresar un texto predefinido en campos de la aplicación. Para el acceso personalizado y otras entradas de texto predefinido, la prueba *Robo* puede ingresar texto en campos `EditText`. Para cada `string`, debes identificar el campo `EditText` con el nombre de un recurso de Android.

La prueba Robo tiene dos métodos mutuamente exclusivos para admitir el acceso:

- **Acceso personalizado:** Si proporcionas las credenciales de una cuenta de prueba, debes indicarle a la prueba *Robo* dónde ingresarlas.

- **Acceso automático:** Si la aplicación tiene una pantalla de acceso que usa una cuenta de Google para la autenticación, la prueba *Robo* usa una cuenta de prueba de Google, a menos que proporciones las credenciales de una cuenta de prueba para el acceso personalizado.

Preguntas de repaso: Firebase Stability

5.6. Servicio de Backup de Google

El servicio de *Backup* de Google te permite realizar una copia de seguridad de los datos de tus aplicaciones en la nube. El fin es proporcionar un punto de restauración de datos y configuraciones de aplicaciones. Si un usuario realiza una restauración de fábrica o se cambia a un nuevo dispositivo con Android, el sistema restaura automáticamente los datos de la copia de seguridad cuando la aplicación se vuelve a instalar. De esta manera, los usuarios no tienen que reintroducir datos de configuración de la aplicación. Este proceso es completamente transparente para el usuario. No afecta a la funcionalidad o la experiencia del usuario en la aplicación. La aplicación necesita un API 8 o superior además de una cuenta de Google. A partir de la API 23 (Android 6.0) el usuario tiene la posibilidad de utilizar un nuevo servicio de Google Play Services, Auto Backup. Realiza una copia de seguridad completa de los datos de la aplicación sin añadir código extra.

Durante una operación de copia de seguridad, el administrador de copia de Android (`BackupManager`) busca en la aplicación los datos de copia de seguridad, y luego se los pasa a un transporte de copia de seguridad, que a su vez los almacena en la nube. Durante una operación de restauración, `BackupManager` recupera los datos de la copia de seguridad y los devuelve a la aplicación. De forma que la aplicación pueda restaurar los datos en el dispositivo. Es posible realizar una solicitud para pedir una restauración, pero eso no debería ser necesario. Android realiza automáticamente la operación de restauración cuando la aplicación es instalada y existen datos de copia de seguridad asociadas con el usuario. El escenario principal en el que los datos de copia de seguridad se restauran es cuando un usuario restablece o actualiza a un nuevo dispositivo y sus aplicaciones previamente instaladas se reinstalan.

El servicio de copia de seguridad no está diseñado para la sincronización de datos de la aplicación con otros clientes o guardar datos que desees acceder durante el ciclo de vida normal de la aplicación. No puedes leer o escribir datos de copia de seguridad a demanda y no se puede acceder a ella de ninguna otra manera que a través de las API proporcionadas por el administrador de copia de seguridad.

El transporte de copia de seguridad es el componente del lado del cliente enmarcado en Android, que es personalizable por el fabricante del dispositivo y del proveedor de servicios. No se garantiza que la copia de seguridad esté disponible en todos los dispositivos con Android. Sin embargo, tu aplicación no se verá afectada en caso de que un dispositivo no proporcione un transporte de copia de

seguridad. Si crees que los usuarios se beneficiarán de la copia de seguridad de datos en la aplicación, entonces puedes desarrollarla, probarla, y luego publicar la aplicación sin preocuparte sobre si los dispositivos no pueden realizar copias de seguridad y causar un error en la aplicación.

Puedes estar seguro de que tus datos de copia de seguridad no pueden ser leídos por otras aplicaciones en el dispositivo.

5.6.1. Fundamentos

En versiones de Android menores a 6.0, para realizar las copias de seguridad de los datos de una aplicación, es necesario implementar un agente de copia de seguridad. Este agente es llamado por el administrador de copia de seguridad para que le proporcione los datos que desea copiar. También es llamado para restaurar la copia de seguridad cuando la aplicación se vuelve a instalar. El administrador de copia de seguridad se ocupa de todas sus transacciones de datos con el almacenamiento en la nube y el agente de copia de seguridad se ocupa de todas las transacciones de datos con la aplicación.

Para implementar un agente de copia de seguridad, debes:

- Declarar el agente de copia de seguridad en el archivo de *manifest*.
- Registrar la aplicación con el servicio de copia de seguridad activado.
- Definir un agente de copia de seguridad mediante `BackupAgent` o `BackupAgentHelper`

La clase `BackupAgent` proporciona la interfaz con la cual la aplicación se comunica con el administrador de copia de seguridad. Si se extiende esta clase directamente, es necesario sobrescribir `onBackup()` y `onRestore()` para realizar y restaurar la copia de seguridad.

La clase `BackupAgentHelper` proporciona una envoltura conveniente para la clase `BackupAgent`, lo que minimiza la cantidad de código que se necesita escribir. En `BackupAgentHelper`, debes utilizar uno o más objetos «ayudantes», que automáticamente realizan y restauran la copia de seguridad de ciertos datos. Por lo tanto, no es necesario implementar `onBackup()` y `onRestore()`.

Android proporciona actualmente ayudantes que realizan copias de seguridad y restauran archivos completos de `SharedPreferences` y archivos de almacenamiento interno.

5.6.2. Auto Backup for Apps

Para aplicaciones cuya versión del target SDK sea Android 6.0 (API 23) o superior y que se ejecuten en dispositivos con Android 6.0 o superior automáticamente se realizará una copia de seguridad de los datos en la nube. El sistema realiza automáticamente la copia de prácticamente todos los datos de la aplicación por defecto, y lo realiza sin tener que escribir código adicional.

El usuario debe activar *Auto Backup for Apps*. El diálogo para realizar la activación aparece en el asistente de configuración o cuando se configura la primera cuenta de Google en el dispositivo.

Cuando un usuario instala la aplicación en un nuevo dispositivo, o la reinstala (por ejemplo, después de una restauración de fábrica), automáticamente el sistema restaura sus datos desde la copia en la nube. Vamos a ver como configurar la característica Auto Backup for Apps, explicando su comportamiento por defecto y como excluir datos que no se quiera que el sistema realice copia de seguridad.

La copia se guarda encriptada en la cuenta Google Drive del usuario. No cuenta en la cuota de Google Drive del usuario. Cada aplicación puede almacenar hasta 25MB. Una vez superado este límite, la aplicación deja de enviar datos a la nube. Si la aplicación solicita una restauración, obtiene los últimos datos enviados a la nube.

La copia se realiza cuando el dispositivo se encuentra en las siguientes condiciones:

- Está en reposo.
- Se está cargando.
- Está conectado a una red Wi-Fi.
- Han transcurrido al menos 24 horas desde la última copia.

Activa la Autobackup en una aplicación añadiendo el artributo `android:allowBackup` con el valor `true` en la etiqueta `Aplicaction` en el manifiesto.

Dependiendo de que datos necesite la aplicación o como los guarde, puede ser necesario utilizar reglas específicas para incluir o excluir ciertos ficheros o directorios. Especificaremos estas reglas a través del manifiesto de la aplicación, donde indicaremos que fichero XML contendrá el esquema de configuración de la copia.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
   package="org.example.eventos">
   <uses-sdk android:minSdkVersion="23"/>
   <uses-sdk android:targetSdkVersion="23"/>
   <application ...
      android:fullBackupContent="@xml/esquema_backup">
   </application>
   ...
</manifest>
```

El atributo `Android:fullBackupContent` indica que fichero XML, que se encuentra en el directorio res/xml, contiene las reglas para controlar que ficheros son copiados. Por ejemplo, si queremos excluir un archivo de la copia, el fichero de configuración sería el siguiente:

```
<?xml version="1.0" encoding="utf-8"?>
<full-backup-content>
   <exclude domain="file" path="device_info.txt"/>
</full-backup-content>
```

Por defecto, la copia automática excluye de la copia ficheros temporales y de cache. Los ficheros afectados son:

- Ficheros de los directorios obtenidos por los métodos `getCacheDir()` y `getCodeCacheDir()`.
- Ficheros localizados en un almacenamiento externo, a menos que residan en el directorio obtenido mediante el método `getExternalFilesDir()`.
- Ficheros del directorio obtenido por el método `getNoBackupFilesDir()`.

Podemos especificar que ficheros incluir o excluir de la copia. La sintaxis del fichero XML de configuración es la siguiente:

```
<full-backup-content>
   <include domain=["file" | "database" | "sharedpref" | "external"
          | "root"]
   path="string" />
   <exclude domain=["file" | "database" | "sharedpref" | "external"
          | "root"]
   path="string" />
</full-backup-content>
```

Los siguientes elementos y atributos te permiten especificar que ficheros incluir o excluir de la copia:

- `<include>`: Especifica el conjunto de recursos a copiar. Si se especifica un elemento `<include>`, el sistema solo hará la copia de este elemento. Puedes especificar múltiples elementos usando varios `<include>`.
- `<exclude>`: Especifica que datos quieres que el sistema excluya cuando se realice la copia. Si se incluyen elementos `<include>` y `<exclude>`, los elementos `<exclude>` tienen preferencia.
- `domain`: Indica que tipo de recursos van a ser incluidos o excluidos de la copia. Puedes asignar los siguientes valores:
  - `root`: Si los recursos se encuentran en el directorio raíz de la aplicación.
  - `file`: Indica un recurso en el directorio retornado por el método `getFilesDir()`.
  - `database`: Especifica la base de datos que retorna `getDatabasePath()` o que interactua con la aplicación mediante la clase `SQLiteOpenHelper`.
  - `sharedpref`: Especifica el objeto `SharedPreferences` que retorna el método `getSharedPreferences()`.

    - `external`: Indica que recurso está en un almacenamiento externo, y corresponde a un fichero que está en el directorio que retorna el método `getExtenalFilesDir()`.
- `path`: Indica la ruta al fichero que quieras incluir o excluir en la copia.

Puedes elegir no realizar la copia automática configurando el atributo `android:allowBackup` a `false` en el elemento `application` en el manifiesto. Veamos un ejemplo:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      xmlns:tools="http://schemas.android.com/tools"
      package="org.example.evento">
   <uses-sdk android:minSdkVersion="23"/>
   <uses-sdk android:targetSdkVersion="23"/>
   <application ...
      android:allowBackup="false">
   </application>
   ...
</manifest>
```

Hay dos escenarios en los cuales puedes necesitar soportar versiones de Android inferiores a Android 6.0 (API 23): actualizando una aplicación para aprovechar la nueva funcionalidad auto backup de Android 6.0, mientras continúas soportando versiones anteriores. O puedes realizar una nueva aplicación, pero quieres asegurarte de que funcionará en dispositivos con versiones anteriores a Android 6.0 además de tener la funcionalidad de auto backup.

Si la aplicación utiliza Android Backup añade el atributo `android:fullBackupOnly="true"` en el elemento `<application/>` en el manifiesto. Cuando se ejecuta en un dispositivo con Android 5.1 (API 22) o inferior, la aplicación realiza la copia con *Android Backup*. Si el dispositivo es Android 6.0 o superior utiliza *Auto Backup for Apps*.

Puedes personalizar el proceso en `onCreate()` o `onFullBackup()`. Por ejemplo, para recibir una notificación cuando ocurra una restauración en `onRestoreFinished()`.

Una vez configurada la copia, deberías probarla para asegurarte que guarda y restaura correctamente. Para ayudar a analizar el fichero XML, activa el *logging* antes de realizar el test:

```
$ adb shell setprop log.tag.BackupXmlParserLogging VERBOSE
```

Primero inicializa `BackupManager` ejecutando:

```
$ adb shell bmgr run
```

Para ejecutar manualmente la copia, ejecuta el siguiente comando. Reemplaza `<PACKAGE>` con el nombre del paquete de la aplicación:

```
$ adb shell bmgr fullbackup <PACKAGE>
```

Para iniciar la restauración, ejecuta. Reemplaza `<PACKAGE>` por el nombre del paquete:

```
$ adb shell bmgr restore <PACKAGE>
```

Puedes restaurar una aplicación desinstalándola y reinstalándola. Los datos son automáticamente restaurados desde la nube cuando la instalación se ha completado.

Si la copia falla, podemos limpiar los datos de la copia y los metadatos asociados desactivando y activando Backup en Ajustes, reseteando a valores de fábrica el dispositivo o ejecutando el siguiente comando:

```
$ adb shell bmgr wipe <TRANSPORT> <PACKAGE>
```

El valor de `<TRANSPORT>` nos lo indica el paquete `com.google.android.gms`. Para obtener la lista de transportes ejecuta:

```
$ adb shell bmgr list transports
```

Para aplicaciones Firebase Cloud Messaging (FCM), copiar el identificador puede causar comportamientos inesperados al restaurarlo. Cuando un usuario instala la aplicación en un dispositivo nuevo, el identificador FCM no es el mismo para el nuevo dispositivo. Excluye el identificador en la copia.

Preguntas de repaso: *Android Backup Service*