[Objetivo]

La prueba consiste en la elaboración de una aplicación (*.apk y código fuente) que responda a los requisitos que a continuación se detallan, que lo haga de forma eficiente, bien diseñada y que esté libre de errores de compilación o en tiempo de ejecución.

Valoraciones

Algunos apartados pueden incluir el término *"Valoraciones*". Esto significa que se tendrá en cuenta implementar dicha condición aunque **no es obligatorio** hacerlo.

Libre

Algunos apartados pueden incluir el término *"Libre"*. Esto significa que **es obligatorio** implementar dicha condición, aunque el desarrollador cuenta con total libertad para hacerlo de la forma que desee.

[Especificaciones]

Descripción general:

❏ Salvo que se indique lo contrario, se puede personalizar o maquetar cada Activity/Fragment como mejor se considere oportuno. Se pueden usar *themes*, elementos nativos de Android UI, o recursos de creación propia.

❏ La aplicación debe ser compatible, al menos, para todas las versiones a partir de *Android 2.3.3 (API-10, GINGERBREAD_MR1)*

Valoraciones generales:

❏ Usar fragments.

❏ Adaptar UI para distintas resoluciones.

❏ Diseñar un spinner (no nativo) para cuando haya peticiones al servidor (excepto en *ProCon.java*, que serán en segundo plano) utilizando las imágenes de la carpeta "Spinner" de los recursos dados.

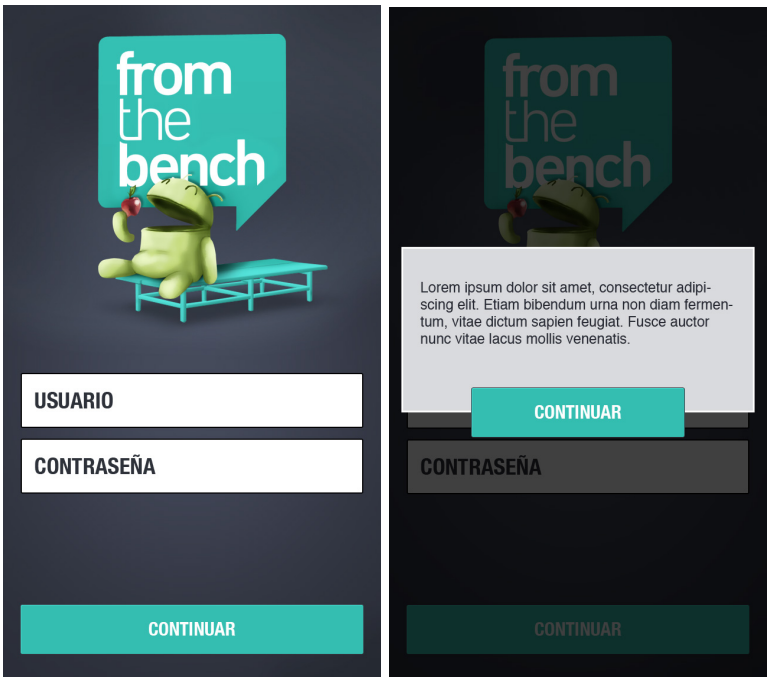
❏ Implementar respuesta (*feedback*) para cuando no existe conexión a Internet.

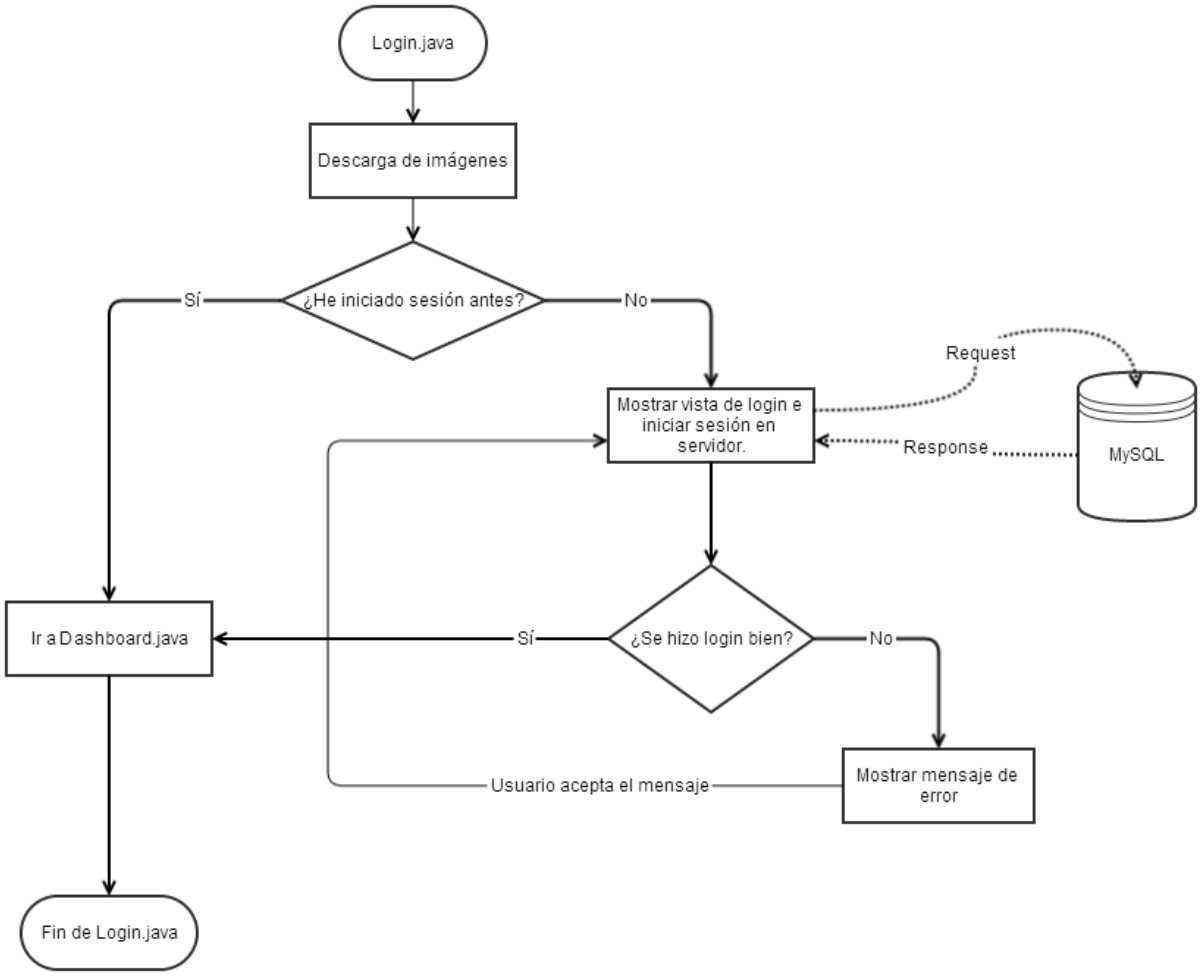
Datos generales:

❏ Para poder hacer la prueba necesitarás un usuario y contraseña. Si no se te ha suministrado ninguno, envía un email a eric.zurera@fromthebenchgames.com

❏ El fichero de recursos gráficos que tendrás que utilizar puedes descargártelo de http://ftbsports.com/android/recursos.

(1) Login.java





Descripción

Login.java actuará a modo de Launcher. Lleva la lógica para decidir si un usuario pertenece a la base de datos y, por tanto, que pueda acceder a *Dashboard.java*. Una vez logueado, el usuario debe poder acceder directamente a *Dashboard.java* sin tener que volver a hacer login si cierra la aplicación.

Una vez se carga *Login.java* hay que iniciar la descarga de imágenes (*DownloadService.java*) explicado más adelante.

Para diseñar la UI utiliza los recursos de la carpeta "*Login*" y *"Login Alert"*.

Valoraciones

❑ Utilizar, a modo de diálogo, un layer personalizado con las imágenes provistas en la carpeta "*Login_alert*" de los recursos dados.

Datos

Llamada para hacer login (GET):

`http://ftbsports.com/android/api/login.php?user=[user]&password=[password]`

Sustituye `[user]` y `[password]` por el nombre de usuario y contraseña suministradas.

Las distintas respuestas son:

Login ok:
{“status”:0}

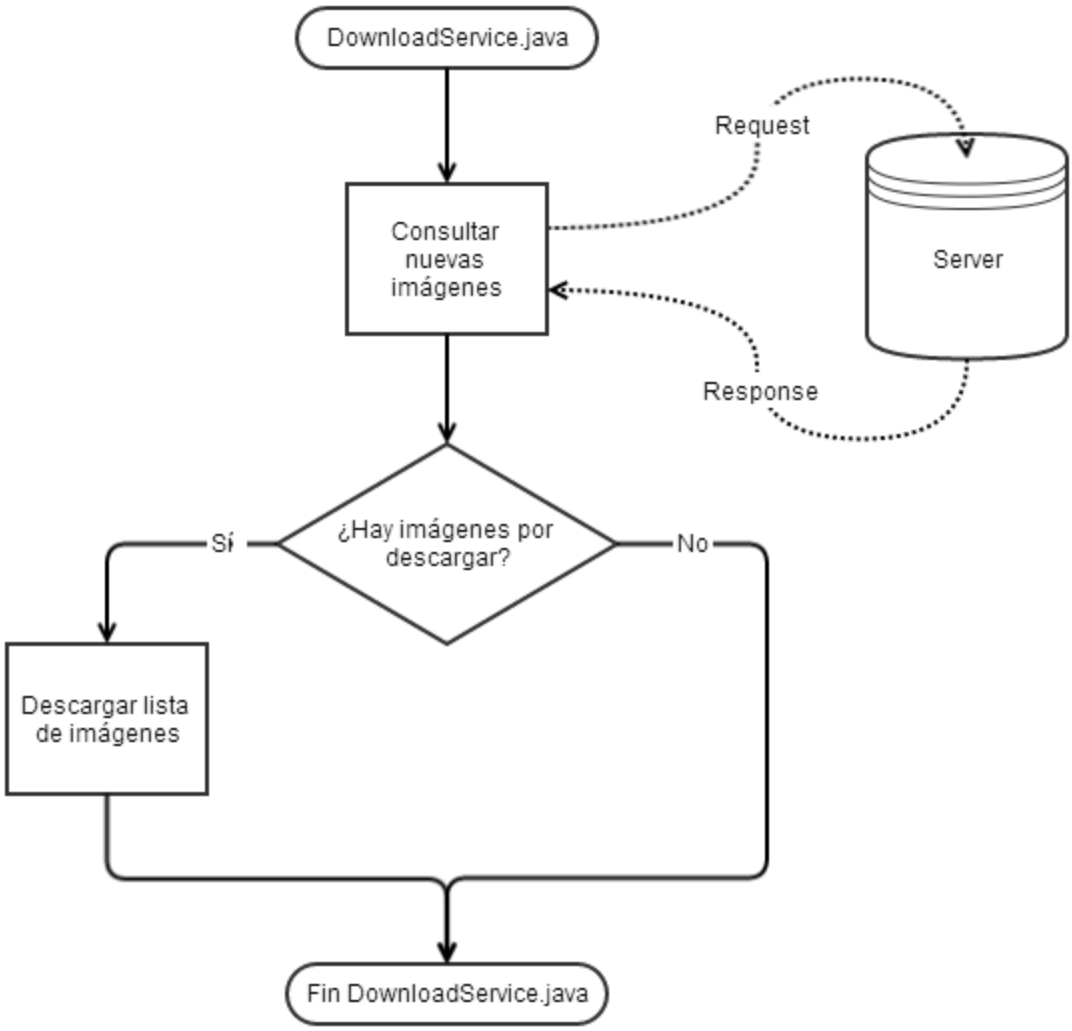
Login ko:

```
{“status”:-1, “message”:”Error en el login. ¿Usuario y contraseña correctos?”}
```

Error desconocido (saldrá de forma aleatoria):

```
{“status”:-2}
```

(1.1) DownloadService.java



Descripción

Utilizar *DownloadService.java* dentro de *Login.java* para descargar una caché de imágenes desde servidor en la memoria del dispositivo (a criterio del desarrollador). Implementarlo mediante un *IntentService*.

Valoraciones

❑ Crear método propio para la descarga de imágenes (sin usar ninguna librería de descarga de imágenes)

Datos

Llamada para comprobar si hay imágenes por descargar (POST):
`http://ftbsports.com/android/api/get_images_cache.php`

Para obtener la lista de imágenes a descargar, es necesario **enviar en cada llamada**, mediante POST, el número de imágenes ya descargadas (almacenadas en el directorio que hayas elegido) y lista de nombres según la siguiente estructura (ejemplo):

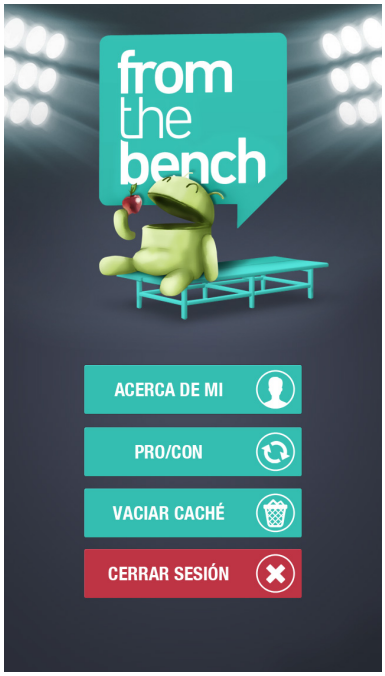
```
data={"count": 3, "list":["imagen1.png", "imagen2.png", "imagen3.png"]}
```

O bien lo siguiente si no se dispone de imágenes:
`data={"count":0, "list":[]}`

Las posibles respuestas son:
a) Si no hay imágenes que descargar:
`{"list":[]}`

b) Si sí hay imágenes por descargar (ejemplo):
`{"list":[{"url":"http://ftbsports.com/android/cdn/imagen1.png","nombre":"imagen1.png"},{"url":"http://ftbsports.com/android/cdn/imagen1.png","nombre":"imagen2.png"},{"url":"http://ftbsports.com/android/cdn/imagen1.png","nombre":"imagen3.png"},{"url":"http://ftbsports.com/android/cdn/imagen1.png","nombre":"imagen4.png"}]}`

(2) Dashboard.java



Descripción

Dashboard.java mostrará un pequeño menú para usuarios logueados. Contará con cuatro botones que llevarán a distintas secciones (explicadas más adelante). Para este apartado, se deberá hacer uso de la animación en Android. Podrá usarse los métodos nativos o agregar cualquier librería externa de animación o compatibilidad (como *nineoldandroids*)

Contará con una lista de botones que nos llevará a los siguiente apartados:

1. Botón "Acerca de mí" abrirá *AboutMe.java*.
2. Botón "Pro/Con" abrirá *ProCon.java*.
3. Botón "Vaciar caché" eliminará la caché de imágenes.
4. Botón "Cerrar sesión" cerrará la sesión activa volviendo a *Login.java*.

Para diseñar la UI utiliza los recursos de la carpeta *"Dashboard"*.

Valoraciones

❑ Procurar que la animación final sea lo más fiel a la indicada.

Datos

A continuación se detallan las distintas animaciones. Seguirá el siguiente formato:

➜ `[t0, td], tipo_animación, nombre_imagen_a_animar`

Donde:

- `t0` son los milisegundos que han pasado desde que ha sido cargada *Dashboard.java*. Implica cuándo debe iniciar la animación.
- `td` son los milisegundos que dura la animación.
- `tipo_animación` es el tipo de animación a utilizar (ejemplos: translación, rotación, escalado,...)
- `nombre_imagen_a_animar` es el nombre de la imagen o recurso que hay que animar

Las animaciones son:

➔ [0, 600], translación de arriba a abajo, logo_ftb.png
➔ [500, 1000], translación de izquierda a derecha, lateral_izq.png
➔ [500, 1000], translación de derecha a izquierda, lateral_der.png

Nota: Puedes ver una aproximación de cómo quedaría la animación en el fichero *"00 - Animacion.gif"*, dentro de la carpeta *"Dashboard"*.

(2.1) AboutMe.java

Descripción

AboutMe.java mostrará una descripción de los datos de contacto del desarrollador (nombre, dirección, teléfono de contacto, enlaces de interés,...)

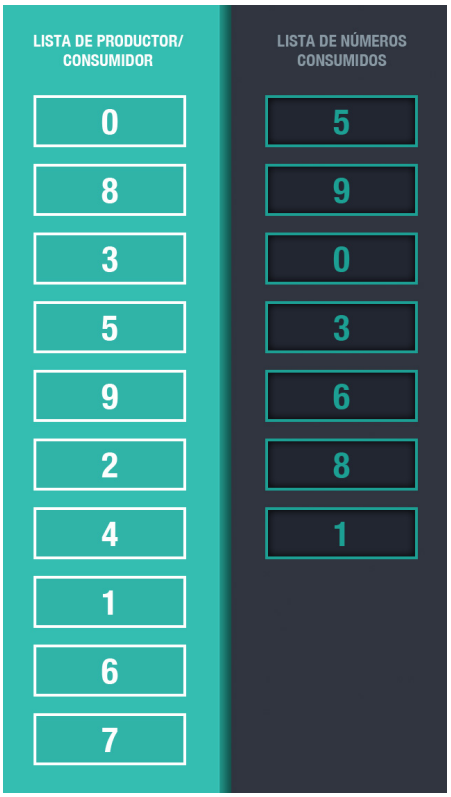
Libre

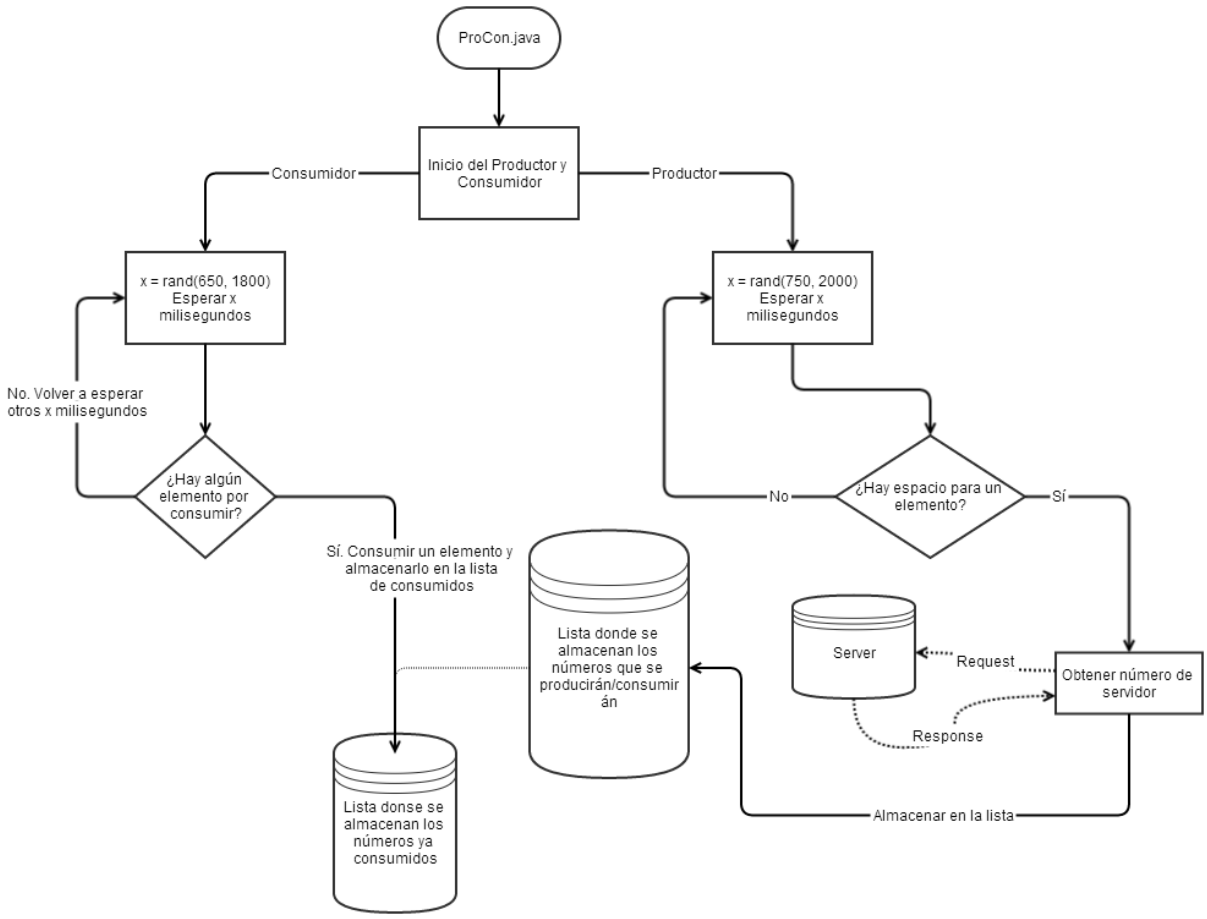
Podrá usarse libremente cualquier elemento (*ListView, ImageView, TextView, custom class,...)* o librería para su desarrollo. Puede incluirse también cualquier otro dato adicional, imagen o recurso, así como diseñar o maquetar como se considere oportuno.

Valoraciones

❑ Se valorará la complejidad y eficiencia del diseño y de la implementación.

(2.2) ProCon.java





Descripción

En esta sección habrá que implementar el clásico ejemplo de productores y consumidores. Para ello se puede hacer uso de la clase *LinkedBlockingQueue<E>* que suministra Java. El productor irá añadiendo números obtenidos mediante una petición GET cada cierto tiempo, mientras que el consumidor los moverá a otra lista también cada cierto tiempo. Hay que considerar que la lista es finita y **sólo podrá almacenar 10 números como máximo**.

Gráficamente habrá dos listas, en vertical. Una hará de lista o cola donde se irán produciendo/consumiendo los números (máx. 10 elementos). La otra lista, de indeterminado tamaño, irá almacenando los números que ya han sido consumidos (usar un *ListView*)

Para diseñar la UI utiliza los recursos de la carpeta *"ProCon"*.

Valoraciones

❏ Optimizar la implementación desarrollada.

Datos

Productor:

1. Hará llamadas al servidor cada `x` segundos (si y sólo si hay sitio para otro elemento). Siendo `x` un número aleatorio entre 750 y 2000 milisegundos.
2. Para obtener el número hacer una petición GET a `http://ftbsports.com/android/api/get_rand.php`. La respuesta será un JSON con la siguiente estructura: `{"num":x}`, siendo x un número entre 0 y 99.

Consumidor:

1. Consumirá un número cada `x` segundos. Siendo `x` un número aleatorio entre 650 y 1800 milisegundos.

(2.3) vaciarCache()

Descripción

vaciarCache() será un método de *Dashboard.java,* asignado al botón "Vaciar caché", cuya única función será eliminar todas las imágenes descargadas, si las hubiera, de *DownloadService.java*.

Valoraciones

❏ Dar feedback al usuario del resultado de la operación.

(2.4) logout()

Descripción

logout() será un método de *Dashboard.java,* asignado al botón "Cerrar sesión", cuya única función será borrar la sesión actual volviendo a *Login.java*.

(3) Consideraciones

Descripción

Intenta responder a las siguientes cuestiones de forma razonada.

1. ¿Qué ventajas y desventajas conlleva utilizar *Fragments* frente a *Activities*?
2. Al hacer login, se ha enviado el usuario y la contraseña como parámetros GET en texto plano.

2.1. ¿Qué inconveniente tiene esto?

2.2. ¿Cómo lo solucionarías? Explica tu respuesta.

3. ¿Es mejor utilizar imágenes en formato *.png o *.jpg?

3.1. ¿Cómo afecta el formato de la imagen a la memoria del dispositivo?

4. Es "frecuente" acceder a un campo de un elemento JSON (*JSONObject* o *JSONArray*) que no existe, provocando un NPE (Null Pointer Exception). ¿Qué ideas se te ocurren para lidiar con este problema?

[Conclusión]

Una vez concluida la prueba:

- ❏ Exporta tu proyecto en formato *.zip. Nómbralo con la siguiente nomenclatura: *nombre_apellido1_apellido2.zip*.
- ❏ Genera la aplicación en formato *.apk con un certificado válido. Nómbrala con la siguiente nomenclatura: *android_nombre_apellido1_apellido2.apk*.
- ❏ Guarda tus respuestas de *"(3) Consideraciones"* en un fichero de texto. Nómbralo con la siguiente nomenclatura: *respuestas_nombre_apellido1_apellido2.txt*.
- ❏ Comprime los tres archivos anteriores en un fichero *.zip con la siguiente nomenclatura: *projecto_nombre_apellido1_apellido2.apk*.
- ❏ Envía el fichero final a: eric.zurera@fromthebenchgames.com.