MONITORIZACIÓN DE PRUEBAS

Título proyecto: ServidorVVS

Validación y Verificación de Software

Integrantes del grupo:

Elena María Delamano Freije
Mateo Fuentes Pombo
Alberto Sueiro Posada

Índice

1. Contexto

En este documento se explicarán las pruebas realizadas en el proyecto "*ServidorVVS*".

Este proyecto emula el comportamiento de una aplicación de reproducción de música.

Los **contenidos** deben tener siempre una duración, un nombre y una lista de reproducción (que explicaremos en detalle más adelante).
Los tipos de contenido son:

- **Canciones**, el contenido simple en el cual se basa nuestro proyecto.
- **Anuncio**, este contenido representa la publicidad. Su duración siempre es de 5 segundos.
- **Emisora**, contenido compuesto por otros tipos de contenido. No tienen duración propia, ya que su duración es la suma de la duración de todos sus contenidos.

Todos los contenidos tienen una lista de reproducción, independientemente de que sean simples (anuncio, canción) o compuestos (emisora).

Si se trata de devolver la lista de reproducción de un contenido simple se devolverá una lista con solo dicho contenido, mientras que si se trata de una emisora, devolverá la lista de los contenidos que la componen (en este caso NO devolverá la propia emisora).

Siguiendo las especificaciones de los contenidos, seguimos con las de los **servidores**:

En el proyecto tenemos 2 tipos de servidores: simple y con respaldo. Su principal diferencia radica en el comportamiento ante una búsqueda vacía. Si un servidor simple realiza una búsqueda que como resultado devuelve una lista vacía, no hace nada más. En cambio, un servidor con respaldo, realizará la misma búsqueda en su servidor asociado.

Ambos servidores pueden introducir anuncios en una búsqueda, siguiendo una serie de especificaciones que se comentarán en el siguiente apartado.

Cuando se da de alta en un servidor, se devuelve un token con 10 usos.

2. Estado actual

- **Alta en un servidor**. Esta funcionalidad representa el login en el servidor. Los usuarios con token válido tendrán ventajas (visualizar contenidos sin anuncios) respecto a los que no estén logueados. El token caduca después de devolver 10 contenidos.
- **Baja en un servidor**. Esta representaría el *logout* en el servidor. Es necesario que se le pase el token que se quiere dar de baja. Si se tratase de usar un token eliminado, lanzaría un error o se trataría como un usuario sin loguearse, según proceda.
- **Agregar contenido en un servidor**. Añade contenido nuevo (de cualquier tipo) al servidor. Sólo un token especial (ej:administrador) puede modificar el contenido del servidor.
- **Eliminar contenido en un servidor**. Busca el contenido y lo elimina del servidor. Sólo un token especial (ej:administrador) puede modificar el contenido del servidor. En caso de no encontrar el contenido, NO lanza error.
- **Buscar contenido en un servidor**. A partir de una cadena y un token dados, busca en los contenidos del servidor todas las coincidencias.

Los servidores pueden devolver anuncios, pero siguen una serie de reglas:

- Una búsqueda con un token válido nunca introducirá anuncios entre los contenidos.
- Si no se le pasa un token (o se le pasa uno inválido) introducirá un anuncio al inicio de la lista y otro por cada 3 contenidos.

En el siguiente apartado se presentan las pruebas realizadas.

3. Registro de pruebas

3.1. Ejecución pruebas dinámicas - JUnit

Con este *framework* se nos ha permitido realizar una ejecución de las clases del proyecto de una manera controlada, de manera que introduciendo una serie de datos de entrada, evaluábamos el valor de retorno esperado.

Son las primeras pruebas que hemos realizado, para saber que la aplicación poseía la funcionalidad esperada.

Los casos de prueba de los contenidos eran bastante sencillos, dado que abarcaban solo las funcionalidades de devolver nombre, duración o lista de reproducción. Y, en el caso de la emisora, agregar o eliminar alguno de sus contenidos.

Debido a la simpleza de estos tests, no los expondremos aquí (*Si se quieren visualizar, acceder aquí*) y nos centraremos en los del servidor. La complicación de esta herramienta estaba en abarcar todos los casos de prueba del servidor:

- Alta de token.
- Baja de token con un token válido - Da de baja el token del servidor.
- Baja de token con un token inválido - Devuelve una excepción de token inválido.
- Buscar contenido con un token válido - Devuelve el contenido.
- Buscar contenido con token invalido - Devuelve una lista vacía, debe devolver el resultado + anuncios.
- Buscar contenido inexistente con token válido - Devuelve lista vacía (en caso de servidor con respaldo, debe buscar en el servidor respaldado).
- Buscar contenido inexistente con token inválido - Devuelve la lista vacía, debe devolver una lista vacía más los anuncios.
- Agregar contenido con token especial.
- Agregar contenido con token inválido (no especial) - Debe devolver un error.
- Eliminar contenido con token especial.

- Eliminar contenido inexistente con token especial - No dice nada, pero debe de mostrar un error.
- Eliminar contenido con token inválido (no especial) - Debe devolver un error de token inválido.
- Eliminar contenido inexistente con token inválido (no especial) - Debe devolver un error de token inválido.

3.2. Pruebas estáticas - FindBugs

FindBugs es una herramienta de análisis estático que permite la búsqueda de posibles errores durante el desarrollo de software.

Para ejecutarlo, hemos utilizado:

- mvn test
- mvn findbugs:check

Esto genera un xml con los bugs y warnings que encuentre en el directorio */target*, con la misma información que la salida por el terminal.

3.3. Validación de la calidad de las pruebas - Cobertura

Cobertura es una herramienta que calcula el porcentaje de código accedido por los tests. Lo hemos utilizado tanto para obtener la cobertura de líneas de códio como la cobertura de ramas.

Para ejecutarlo, hemos utilizado:

- mvn cobertura:cobertura

Esto genera una serie de archivos en la carpeta */target/site/cobertura*. Accediendo al archivo *index.html* podremos ver un análisis de las coberturas de líneas y de ramas de todas las clases del proyecto.

En la primera ejecución que hemos realizado, obtuvimos una cobertura de líneas del 90 %, por lo que nos dimos por satisfechos en la medición y nos centramos en las coberturas de ramas.

La cobertura de ramas no tenía unos resultados tan satisfactorios como la anterior, lo que nos llevó a crear nuevos casos de prueba como solución para que la cobertura aumentase.

3.4. Automatización de la generación de datos en ejecución de pruebas dinámicas - JCheck

Esta herramienta está basada en la librería *QuickCheck* de Haskell.

Para poder ejecutar JCheck necesitamos la librería externa *JCheck.jar* incluida en el directorio */lib* del proyecto.//
Para poder incluirlo como repositorio en maven se debe ejecutar:

mvn install:install-file -Dfile=lib/jcheck.jar -DgroupId=jcheck -DartifactId=jcheck -Dversion=1 -Dpackaging=jar -DgeneratedPom=true

Para realizar las pruebas de JCheck debemos ejecutar con JUnit el archivo: *src/test/java/es/udc/fic/vvs/Practica1/JCheck/ServidorTest.java*

3.5. Pruebas no funcionales - JETM

Para realizar las pruebas no funcionales mediante JETM se debe de ejecutar el archivo:
java -jar src/test/java/es/udc/fic/vvs/Practica1/Jetm/TestJetmServidorSimple.java

Esto imprimirá por terminal una tabla con los resultados.

Un ejemplo del resultado de ejecución puede verse aquí.

3.6. Validación de la calidad de las pruebas - PIT

Es una herramienta para la validación del código que consiste en el llamado "*mutation testing*".

Esto consiste en introducir fallos en el código de manera que se pueda observar cuál es la reacción. En caso de fallar la ejecución, significa que ha sido satisfactoria y el "mutante" muere.

En el caso contrario, el "mutante" vive, lo que nos dice que existe una mala especificación en nuestro código.

4. Registro de errores

4.1. Ejecución pruebas dinámicas - JUnit

Con la realización de las pruebas de JUnit, nos hemos encontrado con una serie de errores de los que no habíamos sido conscientes. Todos estos errores están relacionados con la salida de los métodos, que no devuelven lo esperado.

- En el caso de prueba de "***Buscar contenido con token inválido***", nos encontramos con que en nuestro caso se devuelve una lista vacía, pero el resultado óptimo esperado sería que se devolviese una lista con los contenidos que coinciden con la búsqueda junto con los anuncios que introduce el servidor por no ser un usuario con token válido.
- En el caso de prueba de "***Buscar contenido inexistente con token inválido***" se nos devuelve una lista vacía, pero lo correcto sería que devolviese dicha lista (porque no existe ningún contenido que coincida con la búsqueda) más un anuncio, que coincide con el introducido por el servidor por no haber buscado con un token válido.
- En el caso de prueba de "***Eliminar contenido inexistente con token especial***", nos encontramos con que no dice absolutamente nada. En su lugar, lo apropiado sería que se mostrase un error que nos informara del resultado de la ejecución.

4.2. Pruebas estáticas - FindBugs

En el xml autogenerado en el directorio */target*, nos encontramos con los siguientes errores:

- Salida muerta en el método ***generarToken*** del Servidor de Respaldo.
- Creación de un *String* sin usar el constructor apropiado: ***new String()***.
- Comparación de objetos usando directamente == en lugar del método ***equals*** o ***compareTo***.

Los errores encontrados con esta herramienta ya han sido solventados.

4.3. Validación de la calidad de las pruebas - Cobertura

En la cobertura de ramas, nos hemos encontrado con que no pasaba por algunas. Hemos tenido que incrementar el número de test para contemplar algún caso de prueba que no lo estaba, pero aún así, nos encontramos con algunas condiciones que no fuimos capaces de satisfacer.

5. Estadísticas

No aplica, dado que no hemos llevado un seguimiento exhaustivo.

6. Otros aspectos de interés

6.1. Validación de la calidad de las pruebas - Cobertura

En la cobertura de ramas todavía tenemos muchas sin ejecutarse. Algunas intuimos que son problemas/bugs del plugin, como indicarnos que no pasan por algunas líneas comentadas).

6.2. Validación de la calidad de las pruebas - PIT

En la ejecución de las pruebas de mutación con ***PIT*** no se ha logrado matar a todos los "mutantes" creados.

Hemos conseguido identificar los siguientes casos erróneos, pero no encontramos una solución:

- Prints por pantalla.
- Return de objetos, como por ejemplo token, lo que le lleva a lanzar una excepción runtime. Creemos que esto se debe a que no controlamos el caso en el que el objeto devuelto sea nulo.
- Bucles que no consigue terminar, dando error de *time_out*.