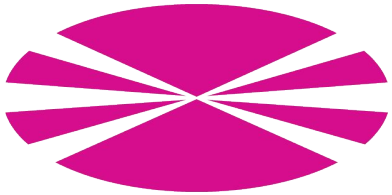


UNIVERSIDADE DA CORUÑA

Testing

Ángel Castro Yáñez <angel.castro2@udc.es>
Pedro Fernández Luces <pedro.fernandez.luces@udc.es>
Avelino Ríos Sáez <avelino.rios@udc.es>

Primeros fallos

Hemos utilizado JUnit para realizar unas primeras comprobaciones de nuestro código y hemos detectado los siguientes problemas:

- No habíamos inicializado la lista de contenidos en Anuncio.
- Recogíamos mal los contenidos en el servidor.

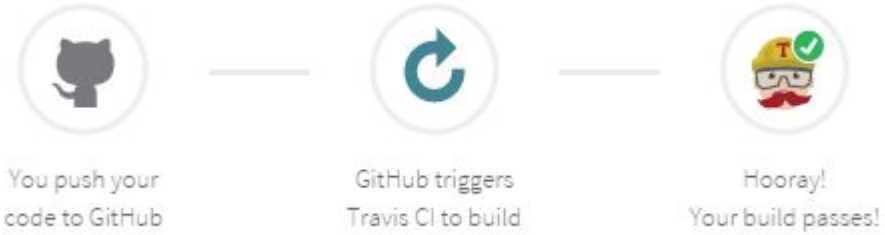
Issues a otros y pull request

- Fallos en los tokens en los servidores de respaldo.
- Error método buscar() en servidor simple.
- Método Buscar() no implementado.
- Hemos realizado un pull request con correcciones a otro grupo.

Issues abiertos a nuestro repositorio

- Número de colaboradores
- Falta documentación.
- obtenerListaReproduccion() en Contenidos de carácter individual.
- Duración al eliminar contenido en Emisora.
- Fallo en ServidorConRespaldo con token inválido y lista vacía.
- No existe Token especial o de administrador.
- Participación desequilibrada.

Travis CI



Con herramientas de integración contínua como Travis nos ayudan a realizar pruebas automáticas de un proyecto para así poder detectar los fallos cuanto antes.

Hemos realizado pruebas de unidad con JUnit de las clases:

- Canción
- Anuncio
- Emisora
- Token

También hemos realizado pruebas de integración a las clases:

- ServidorBasico
- ServidorConRespaldo

COVERALLS

coverage 96%

92.31 91.61 90.91 87.94 89.13 88.41 86.96 86.23 86.43 86.43

86.96 95.92 95.92 95.92 95.92 95.92 95.92 95.92 95.92 95.92

Con esta herramienta hemos mejorado la cobertura de nuestros tests.

Debido a que algunas partes del código generan elementos aleatorios, no es posible realizar un test.

Para utilizar coveralls debemos introducir el plugin en el pom de nuestro proyecto.

```
<plugin>
    <groupId>org.eluder.coveralls</groupId>
    <artifactId>coveralls-maven-plugin</artifactId>
    <version>4.0.0</version>
</plugin>
```

QuickChek®

Get fresh. Go fast.®

- Quickcheck probar las propiedades que deberían de cumplir las funciones, es decir, cada función tiene propiedades deseables lo que se logra con QuickCheck es ver si se cumplen total o parcialmente estas propiedades.
- Una ventaja notoria es que la propiedad es probada con una gran cantidad de casos generados aleatoriamente.

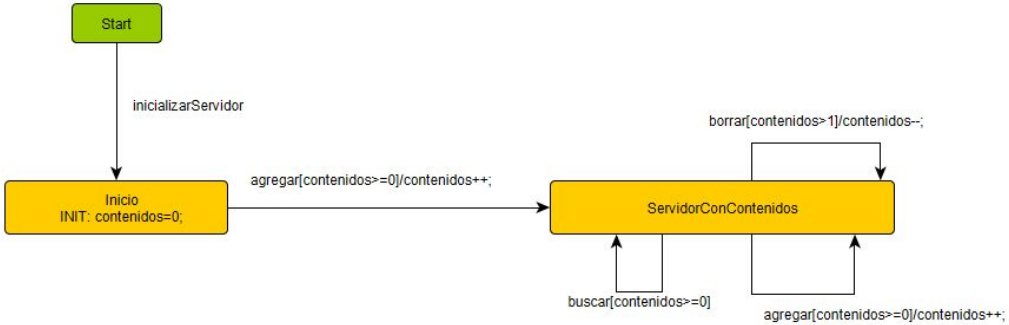
```
public class IntegerGenerator implements Generator<Integer>{


    private final Generator<Integer> gen = PrimitiveGenerators.integers();

    /**
     *
     * @return Integer generado
     */
    public Integer next() {
        return new Integer(gen.next());
    }

}
```

GraphWalker
The Open Source Model-Based Testing Tool



Utilizamos GraphWalker para realizar pruebas dinámicas y pruebas de carga contra nuestro modelo. Dando como resultado que no tenemos pérdidas de memoria y certificando que nuestro programa funciona bien bajo una elevada carga.

- Hemos empleado esta herramienta para realizar pruebas unitarias sobre los servidores, ya que resultaría imposible con JUnit porque los servidores utilizan las clases de Contenido, Anuncio, Emisora, Canción, Token.

- Utilizamos PowerMock para mockear la clase estática *Token.*

- Gracias a esta herramienta hemos encontrado un error en el constructor de servidor.

Pit Testing

Hemos realizado pruebas de mutación para mejorar la calidad de nuestros tests detectando problemas como:

- El test eliminar contenido de emisora no comprobamos que la nueva duración se decrementa.
- El test buscar contenido en servidor, no comprobamos el orden en el que se insertan los anuncios.
- Varios errores en condiciones if y for (Ej: El test era correcto tanto si una condición de un if era (a > 0) como (a >= 0)

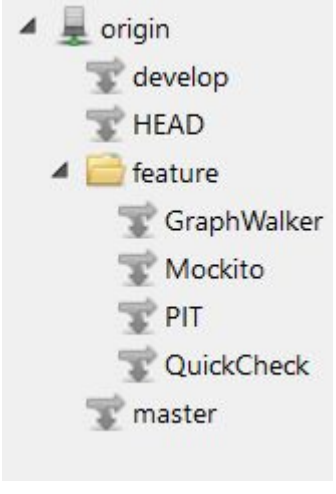
- Búsqueda de variables muertas, llamadas ineficientes, modificadores de las variables sean correctos…

- Comprobamos estilo del código sea correcto

- Prácticas cuestionables a la hora de codificar

Uso repositorio

GitHub




"That's all Folks!"