

Universidad Rey Juan Carlos

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

GRADO EN INGENIERÍA DEL SOFTWARE
2017-2018

Trabajo de Fin de Grado

TestLogAnalyzer: Análisis y comparación de logs de tests

Autor: Carlos Vázquez Losada

Tutor: Micael Gallego Carrillo

Esta memoria forma parte del Trabajo de Fin de Carrera de la titulación de Ingeniería del Software en la Escuela Técnica Superior de Ingeniería Informática de la Universidad Rey Juan Carlos de Madrid, realizado por Carlos Vázquez Losada en 2018.

Se otorga permiso para copiar, distribuir y/o modificar el código de este proyecto bajo los términos proporcionados por la licencia de Software Libre Apache 2.0¹.



Esta obra se encuentra sujeta a la licencia Creative Commons 3.0², que permite compartir y adaptar el contenido de esta obra, pero requiere el reconocimiento de la autoría del mismo.

¹[https:// www.apache.org/licenses/LICENSE-2.0.html](https://www.apache.org/licenses/LICENSE-2.0.html)

²<https://creativecommons.org/licenses/by/3.0/es/>

RESUMEN

En el mundo contemporáneo, las empresas compiten entre sí para ofrecer un mejor servicio a sus clientes que sus competidoras. Ante esta situación, las grandes compañías elaboran software novedoso, utilizando algoritmos, técnicas y herramientas nunca antes conocidas, buscando desmarcarse de sus émulas. Es por este motivo por el que se requieren herramientas que aseguren una correcta implementación, sin fallos, de este software.

Para asegurar un desarrollo sin fallos y cuyo nuevo código realice las funciones para las que ha sido programado, es necesaria la elaboración de tests. Los hay de muchos tipos: de integración, de back end... Todos y cada uno de ellos sirven en determinadas situaciones y tienen fines muy concretos, que los profesionales han de seleccionar dependiendo del proyecto. La información devuelta por estos tests es de gran valor y no sólo deberíamos fijarnos en el registro lingüístico devuelto por los logs, sino en la ejecución como un todo.

TestLogAnalyzer compara los logs generados durante la ejecución de tests entre diferentes versiones del mismo software. Esta funcionalidad es especialmente útil cuando se produce una regresión y se compara el log de la ejecución del test con fallo con las ejecuciones previas de ese mismo log sin fallo.

Palabras clave: Análisis y comparación de logs; Integración continua; ElasTest

DEDICATORIA

Deseo expresar mi agradecimiento a todos los que me han acompañado y ayudado durante la elaboración de este trabajo, particularmente a mis padres Isabel Losada Balles-
ta y Fco. Javier Vázquez de Pablo, sin cuyo apoyo no habría sido posible, y a los que dedico esta obra. Mi agradecimiento asimismo a los profesores que me han aportado el conocimiento que poseo, en especial a Micael Gallego, mi tutor y gran ejemplo a seguir, y a la Biblioteca de la Universidad Rey Juan Carlos por las orientaciones recibidas.

ÍNDICE GENERAL

1. INTRODUCCIÓN.	1
1.1. Motivación del trabajo	1
1.2. Objetivos	3
2. TECNOLOGÍAS, HERRAMIENTAS Y METODOLOGÍAS.	4
2.1. Tecnologías	4
2.1.1. TypeScript	4
2.1.2. Node.js	8
2.1.3. Java	9
2.2. Herramientas	10
2.2.1. Control de versiones	10
2.2.2. Gestión de dependencias	11
2.2.3. Entornos de desarrollo	12
2.2.4. Entornos de integración continua	13
2.2.5. Bases de Datos	15
2.2.6. Despliegue	16
2.2.7. Otras herramientas	17
2.3. Metodología	19
3. MANUAL DE USO.	20
3.1. Visualización de los proyectos	20
3.2. Creación de proyectos y subida de logs	20
3.3. Visualización de un proyecto	22
3.4. Visualización y comparación de una ejecución	23
4. DESCRIPCIÓN INFORMÁTICA	28
4.1. Desarrollo iterativo e incremental	28
4.1.1. Fase 1: Almacenamiento y visualización de la información	28
4.1.2. Fase 2: Análisis de la información	31
4.1.3. Fase 3: Producto autónomo	32
4.2. Requisitos.	33

4.3. Arquitectura	34
4.3.1. Front end	34
4.3.2. Back end	35
4.4. Implementación	36
4.4.1. Algoritmo de comparación	36
4.4.2. Diferenciación por petición a la API REST	38
4.4.3. Tratamiento de múltiples ficheros (.txt y .xml)	39
4.4.4. Integración continua sobre la aplicación	40
4.5. Pruebas	41
4.5.1. Implementación	41
4.5.2. Automatización.	43
5. CONCLUSIONES	44
5.1. Objetivos cumplidos	44
5.2. Líneas futuras de trabajo.	44
BIBLIOGRAFÍA	45

ÍNDICE DE FIGURAS

1.1	Costes económicos asociados a los fallos en el software	1
1.2	Logo de ElasTest	3
2.1	Logo de TypeScript	4
2.2	Logo de Angular	5
2.3	Logo de Angular CLI	5
2.4	Logo de Angular Material	6
2.5	Logo de Teradata Covalent	7
2.6	Logo de NG Bootstrap	7
2.7	Logo de Node.js	8
2.8	Logo de Java	9
2.9	Logo de Spring Boot	9
2.10	Logo de Git	10
2.11	Logo de Tidelift	11
2.12	Logo de npm	11
2.13	Logo de Maven	12
2.14	Logo de WebStorm	13
2.15	Logo de Spring Tool Suite	13
2.16	Logo de Travis CI	14
2.17	Logo de CircleCI	14
2.18	Logo de ELK Stack	15
2.19	Logo de Docker	16
2.20	Logo de Docker Compose	17
2.21	Logo de Postman	18
2.22	Logo de Compodoc	18
2.23	Esquema del desarrollo iterativo e incremental	19
3.1	Página principal de la sexta versión del TestLogAnalyzer	20
3.2	Página de creación de la sexta versión del TestLogAnalyzer	21

3.3	Página de visualización de proyecto de la sexta versión del TestLogAnalyzer	22
3.4	Modo de visualización completa	24
3.5	Modo de visualización sin mensajes de Maven	24
3.6	Modo de visualización de los tests	25
3.7	Modo de visualización de los tests fallidos	25
3.8	Modo de comparación completa con visualización completa	26
3.9	Modo de comparación sin timestamp con visualización completa	26
3.10	Modo de comparación con diferenciación temporal y visualización completa	27
4.1	Diagrama arquitectónico de TestLogAnalyzer	28
4.2	Versión beta del TestLogAnalyzer	29
4.3	Primera versión del TestLogAnalyzer	29
4.4	Página principal de la segunda versión del TestLogAnalyzer	30
4.5	Página de comparaciones de la segunda versión del TestLogAnalyzer . . .	31
4.6	Página de comparaciones de la tercera versión del TestLogAnalyzer . . .	31
4.7	Página principal de la cuarta versión del TestLogAnalyzer	32
4.8	Diagrama del front end	35
4.9	Diagrama del back end	35
4.10	Administración de las ramas del repositorio	43

ÍNDICE DE TABLAS

4.1	Requisitos de la última iteración	34
4.2	Atributos del algoritmo Diff, Match and Patch	37

1. INTRODUCCIÓN

1.1. Motivación del trabajo

Hoy en día, Internet está repleto de servicios y aplicaciones que nos hacen la vida más fácil prácticamente en todos los ámbitos de la cotidianidad. Las empresas compiten día a día por ofrecer un servicio mejor que el que pueden ofrecer sus competidoras a los potenciales usuarios y clientes.

Ante esta situación, y debido a la creciente e increíblemente rápida evolución del sector tecnológico (lo que implica una evolución en muchos ámbitos, incluyendo el software, objetivo de este TFG), resulta imprescindible adaptarse a estos cambios y avances pioneros, ya que mejoran la eficiencia de nuestro código y sus algoritmos, proveyendo servicios más competitivos y con una respuesta más rápida frente a los servicios con tecnologías tradicionales y desactualizadas.

La evolución en nuestro software implica una necesaria e inevitable modificación en el código asociado, con lo que suelen aparecer una gran cantidad de errores, algunos de ellos imperceptibles, y que salen a relucir cuando hemos avanzado en el desarrollo de nuestro proyecto, convirtiendo una labor de minutos en una de horas, e, incluso, días. Además, a este coste temporal hemos de sumarle el coste económico que supone el esfuerzo necesario para solventar estos errores.

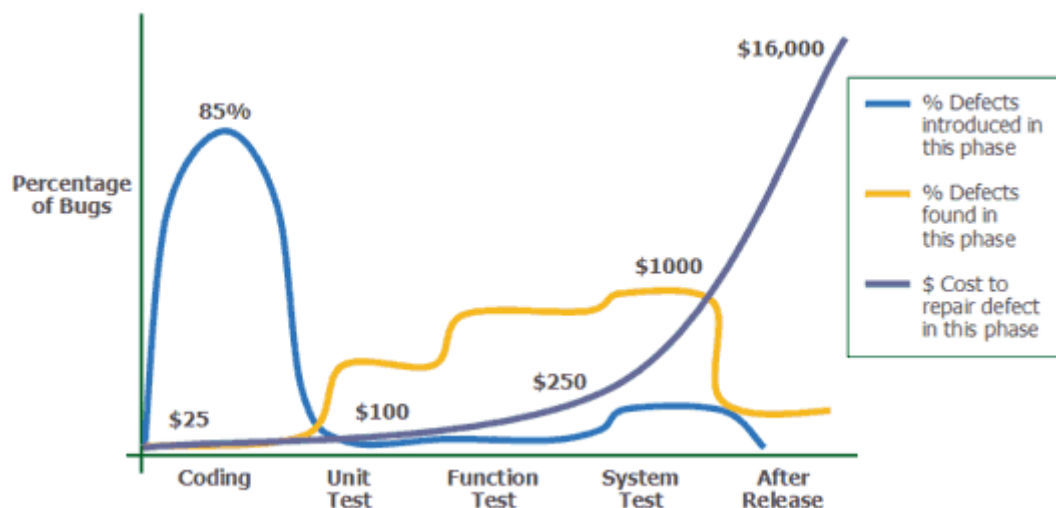


Fig. 1.1. Costes económicos asociados a los fallos en el software

En 1996, Capers Jones [1] hacía explícito, tal y como se muestra en Fig. 1.1, un aumento en el coste de solucionar ciertos bugs no resueltos durante el proceso de desarrollo del software, haciéndose altamente costosos una vez éste ha sido lanzado al mercado.

Como C. Jones mencionó, incluso programando tests, hay un pequeño porcentaje de errores que se escapan a nuestros ojos, por lo que resulta de gran utilidad que desarrolladores y administradores cuenten con herramientas que examinen y revisen la información generada por estos tests en busca de fallos o anomalías.

Prácticamente a diario aparecen errores en el software que, en mayor o menor medida, se traducen en pérdidas económicas para la empresa afectada (ya sea por imposibilidad de que los clientes puedan realizar transacciones económicas, la pérdida de información confidencial de los usuarios, pérdida de clientes...). Revistas como Scientific American [2] hicieron públicas algunas de las pérdidas económicas por fallos en el software más grandes de la historia:

- **AT&T publica su servicio de larga distancia.** Durante 9 horas, el software que controlaba los conmutadores de relevo de larga distancia de esta compañía había sido actualizado con errores al lanzar este nuevo servicio, por lo que, en enero de 1990, cualquier cliente que no fuese de la compañía telefónica AT&T podía realizar una de estas llamadas de larga distancia. AT&T terminó perdiendo más de 60 millones de euros en cargos ese día.
- **El MCO (Mars Climate Orbiter) se desintegra en el espacio.** En diciembre de 1998, el software en las máquinas en tierra que controlaban el sistema de propulsión de la sonda utilizaba las unidades incorrectas (libras por segundo en lugar de Newton por segundo), produciendo que, la sonda espacial robótica de 65 millones de euros ardiera en la atmósfera superior de Marte al impactar en el ángulo incorrecto.
- **Apple Maps direcciona a ningún lugar.** En 2012, tratando de rivalizar con la famosa Google Maps Application, Apple intentó sustituir al aclamado Maps sustituyéndolo por un nuevo mapa incorporado en los nuevos iPhone y creado por ellos mismos. El problema fue que todos los lagos, estaciones, puentes y atracciones turísticas se encontraban mal posicionados o ausentes del mapa.

El testing automático es la mejor forma de controlar que un software hace lo que se espera de él. Una prueba puede terminar de forma satisfactoria o fallar. Cuando un test falla, es necesario buscar el motivo por el que ha fallado. En esta situación, pueden darse dos situaciones: que la prueba haya fallado porque nunca se ha ejecutado como exitosa (en todas las ejecuciones falló el mismo test), en cuyo caso no existen referencias de ejecuciones previas, o bien, el caso de que un test haya finalizado sin errores en ejecuciones previas y, ante un cambio, haya finalizado con error. En este caso, una herramienta como TestLogAnalyzer puede ser muy útil, ya que permite comparar los logs de la ejecución exitosa de un test frente a la ejecución no exitosa del mismo.

Los cambios en el comportamiento de un sistema se manifiestan como cambios en los logs, y ese cambio en los logs puede ayudar a descubrir qué cambio de comportamiento ha hecho que el test haya fallado.

El entorno de integración continua es una plataforma centralizada que permite a un equipo de desarrollo ejecutar tests y es aquí donde una herramienta de este tipo tiene sentido. Es por este motivo por el que TestLogAnalyzer se ha desarrollado como una aplicación web.

1.2. Objetivos

El objetivo de este trabajo es realizar una aplicación que ayude a descubrir de forma más rápida las regresiones ocasionadas en el desarrollo software. Una regresión es un motivo por el que un test que antes se ejecutaba correctamente, lo haya dejado de hacer. Para ello, utiliza los datos que los logs pueden ofrecer en el ciclo de vida de un proyecto.

TestLogAnalyzer³ cubre la necesidad de analizar los logs provenientes de los distintos subsistemas de una aplicación. Permitirá crear proyectos, asociar a los mismos distintas ejecuciones compuestas por logs, los cuáles se analizarán y/o visualizarán según el usuario indique.

ElasTest⁴ (Fig. 1.2) es una plataforma orientada a probar aplicaciones distribuidas y E2E. Esta plataforma proporciona facilidades para el desarrollo de procesos y el acceso a las utilidades propias de pruebas de E2E. También proporciona herramientas para mostrar y analizar logs y métricas de todos los elementos involucrados en estas pruebas. Como futuro módulo de ElasTest, ampliará la funcionalidad que ofrece a sus usuarios.



Fig. 1.2. Logo de ElasTest

TestLogAnalyzer debe, por tanto, comparar logs de ejecuciones de tests atendiendo a su estructura determinada. Debido a que los logs tienen una estructura muy determinada, no es útil recurrir a una comparación normal de texto plano. Por ejemplo, los logs suelen estar dotados de una fecha de ejecución, y esta fecha de ejecución es un valor que cambia con cada línea, por lo que debería de ser un elemento a no tener en cuenta en la comparación, en busca de una precisión absoluta.

³<https://github.com/cvazquezlos/TestLogAnalyzer>

⁴<https://elastest.io/>

2. TECNOLOGÍAS, HERRAMIENTAS Y METODOLOGÍAS

2.1. Tecnologías

Tal y como se ha mencionado con anterioridad, contemplar TestLogAnalyzer como una aplicación web indica que debería tener un front end y un back end. Aprovechamos los conocimientos obtenidos en Desarrollo de Aplicaciones Web, impartida en el Grado en Ingeniería del Software, relacionados con las tecnologías actuales de front end y back end y su desarrollo desde cero.

2.1.1. TypeScript



Fig. 2.1. Logo de TypeScript

TypeScript⁵ (Fig. 2.1) se define a sí mismo como [3] “Es JavaScript evolucionado. TypeScript es un supertipo de JavaScript que se compila como JavaScript en cualquier navegador, host o Sistema Operativo.”. Parte de JavaScript⁶, añadiendo el tipado estático y el concepto de objeto basado en clases. Extiende la sintaxis de JavaScript, por lo que cualquier código de éste debe funcionar sin problemas en TypeScript.

Con el tipado estático, los tipos son opcionales debido a la inferencia, y permiten definir interfaces entre los componentes del software de cara a obtener información sobre el comportamiento de las bibliotecas de JavaScript existentes.

TestLogAnalyzer será una aplicación web SPA con la parte del front end implementada en Angular 5 con TypeScript. En cuanto a la parte del back end, será desarrollada en Spring con Java. En adición a lo anterior, estará dotada de una base de datos Elasticsearch.

⁵<https://www.typescriptlang.org/>

⁶<https://www.javascript.com/>

Angular



Fig. 2.2. Logo de Angular

Angular⁷ (Fig. 2.2) es un framework para aplicaciones web desarrollado en TypeScript, de código abierto y mantenido por Google, que se utiliza para la creación de aplicaciones web SPA y basado en el clásico patrón MVC.

Además, Angular basa su comportamiento en el concepto de componente. Los componentes pueden ser vistos como pequeñas partes de una interfaz que son independientes entre sí, favoreciendo la reusabilidad, la legibilidad del código y su mantenibilidad.

Sin embargo, [4] cabe mencionar que, para programadores poco experimentados, Angular puede ser un mal punto de partida, dado que es engorroso y complejo. En cualquier caso, la activa comunidad, la inyección de dependencias y la escalabilidad hacen que su utilización sea una buena alternativa.

Angular CLI



Fig. 2.3. Logo de Angular CLI

⁷<https://angular.io/>

Angular CLI⁸ (Fig. 2.3) es una herramienta que permite generar aplicaciones Angular, además de facilitar su desarrollo y mantenimiento.

La creación de proyectos Angular puede ser realizada sin el empleo de esta herramienta, de forma manual, pero el coste del tiempo requerido en comparación con la rapidez del CLI hace que sea indispensable si se quiere tener un proyecto establecido y ejecutable en unos minutos.

De entre las ventajas más importantes de esta herramienta es que es ideal e indispensable para principiantes, incluyendo un webpack y numerosas herramientas de testing tales como Karma⁹, Jasmine¹⁰ o e2e¹¹. Las dependencias de un proyecto Angular también serán administradas por esta interfaz.

Angular Material



Fig. 2.4. Logo de Angular Material

Angular Material¹² (Fig. 2.4) es la adaptación del Material Design de Google que provee a las aplicaciones desarrolladas en Angular diferentes componentes, rápidos, componentes y versátiles que facilitan que las aplicaciones Angular sigan las directrices de este estilo.

El Material Design¹³ busca crear un lenguaje visual que combine los principios de un buen diseño y las posibilidades que traen las nuevas tecnologías. La consecuencia de esta visión es un sistema que posibilita una experiencia uniforme en diferentes plataformas y dispositivos.

⁸<https://cli.angular.io/>

⁹<https://karma-runner.github.io/2.0/>

¹⁰<https://jasmine.github.io/>

¹¹<https://github.com/angular/protractor/>

¹²<https://material.angular.io/>

¹³<https://material.io/design/>

Teradata Covalent



Fig. 2.5. Logo de Teradata Covalent

Teradata Covalent¹⁴ (Fig. 2.5) es una plataforma para interfaces gráficas de código abierto elaborado y mantenido por Teradata que combina un lenguaje de diseño con Angular y Angular Material.

Una de las características más importantes de este framework es que está construido completamente con Angular, concretamente en la versión 4. Debido a este motivo y a que esta plataforma evoluciona más lentamente que el propio Angular, algunos componentes para la aplicación han sido cogidos directamente del Angular Material. Además, sigue el patrón establecido por el Material Design, por lo que ambos estilos de diseño son completamente compatibles, otorgando un número más grande de posibilidades gráficas.

NG Bootstrap



Fig. 2.6. Logo de NG Bootstrap

¹⁴<https://teradata.github.io/covalent/>

NG Bootstrap¹⁵ (Fig. 2.6) es la versión de Bootstrap 4 para Angular. Esta librería ha sido construida desde cero por el equipo de Bootstrap para evitar la problemática utilización del Bootstrap original con el problemático JavaScript que tanto rechazo provoca en las aplicaciones Angular. En su lugar, esta librería ha sido desarrollada casi por completo en TypeScript y su instalación se produce en dos sencillos pasos.

NG Bootstrap es nativo de Angular y no requiere de dependencias de terceros en JavaScript. Todos los widgets son accesibles y están compuestos por elementos HTML y atributos apropiados y la navegación por teclado y el foco funcionan como el usuario puede esperar.

Se ha utilizado esta librería debido a un componente que considerábamos útil y que las otras librerías descritas no ofrecían. Éste es el popover.

2.1.2. Node.js



Fig. 2.7. Logo de Node.js

Node.js¹⁶ (Fig. 2.7) es un entorno de ejecución para JavaScript construido con el motor de JavaScript V8 de Chrome. Utiliza un modelo de operaciones I/O sin bloqueo y orientado a eventos, de ahí su gran eficiencia.

En cuanto al porqué de nuestra elección para utilizarlo podemos encontrar aspectos como:

- **Concurrencia.** [5] Node.js funciona con un modelo de evaluación de un único hilo de ejecución, usando entradas y salidas asíncronas que pueden ejecutarse concurrentemente en un número de hasta cientos de miles sin incurrir en costos asociados al cambio de contexto.
- **V8**¹⁷. Entorno de ejecución para JavaScript de Google Chrome. Escrito en C++, compila el código JavaScript en máquina en lugar de interpretarlo en tiempo real.

¹⁵<https://ng-bootstrap.github.io/>

¹⁶<https://nodejs.org/es/>

¹⁷<https://developers.google.com/v8/>

2.1.3. Java

Como es lógico, todo front end debe tener un back end de la que poder extraer la información que va a ser mostrada en esa interfaz gráfica.

Hay una enorme cantidad de posibilidades en cuanto al desarrollo de entornos back ends con una gran variedad de lenguajes en los que ser implementados, pero, basándonos en la experiencia ya adquirida en la asignatura Desarrollo de Aplicaciones Web, decidimos inclinarnos por Spring, una plataforma basada en Java.



Fig. 2.8. Logo de Java

Java¹⁸ (Fig. 2.8) es un lenguaje de programación de propósito general, concurrente y orientado a objetos. Con una sintaxis derivada de C++ , a diferencia de éste, Java está completamente orientado a objetos, siendo, prácticamente todo, un objeto.

Un objeto [6] es una unidad dentro de un programa computacional que consta de un estado y de un comportamiento, y que contiene datos almacenados y tareas realizables durante un tiempo de ejecución determinado. Un objeto se crea mediante la instanciación de la clase a la que pertenece.

Spring Boot



Fig. 2.9. Logo de Spring Boot

¹⁸<https://www.java.com/es/>

Spring Boot¹⁹ (Fig. 2.9) es un módulo de Spring que proporciona todo lo necesario para la creación de una aplicación de forma muy sencilla, similar a Angular CLI para Angular visto anteriormente.

Permite crear proyectos con un mínimo de configuración e independientes entre sí, permitiendo incrustar tecnologías como Tomcat²⁰, Jetty²¹ o Undertow²² sin realizar un previo despliegue. Además, proporciona un archivo de configuración (conocido como POM) inicial, de tal forma que libra al desarrollador de la engorrosa tarea de redactar el suyo propio desde cero (permite un grado de personalización inicial bastante aceptable).

2.2. Herramientas

Las herramientas permiten utilizar las tecnologías anteriormente mencionadas para solidificar los conceptos en algo completamente ejecutable y funcional. Han sido minuciosamente seleccionadas para lograr la máxima eficiencia y productividad en la creación del proyecto.

2.2.1. Control de versiones

A pesar de realizar el proyecto de forma individual, era imprescindible contar con un control de versiones a través de las cuáles, un usuario ajeno al proyecto pueda visualizar la evolución de una idea en un proyecto, y las distintas fases de este.

Git



Fig. 2.10. Logo de Git

Git²³ (Fig. 2.10) es un sistema de control de versiones gratis y de código abierto diseñado para administrar tanto proyectos grandes como pequeños con rapidez y eficiencia.

Git cuenta con una ristra de comandos que pueden ser aplicados para crear, eliminar o modificar directorios y archivos, o para subir o descargar cambios...

¹⁹<https://spring.io/projects/spring-boot>

²⁰<http://tomcat.apache.org/>

²¹<https://www.eclipse.org/jetty/>

²²<http://undertow.io/>

²³<https://git-scm.com/>

En concreto, el proyecto ha sido realizado empleando GitHub, que no es más que el Git original llevado a un entorno visual mediante la elaboración de una excelente y rápida aplicación SPA. Además, Git cuenta con una aplicación de escritorio que ha sido utilizada para facilitar el proceso de desarrollo, GitHub Desktop²⁴.

2.2.2. Gestión de dependencias

Una dependencia [7] es una aplicación o una biblioteca requerida por otro programa para poder funcionar correctamente. Por ello se dice que dicho programa depende de tal aplicación o de tal biblioteca.

Tidelift



Fig. 2.11. Logo de Tidelift

Tidelift²⁵ (Fig. 2.11), o anteriormente conocido como Dependency CI, fue empleado en los primeros meses de implementación del proyecto hasta que se convirtió en una herramienta de pago mediante suscripción para empresas.

Esta herramienta examinaba las dependencias de todo el proyecto, reconociendo automáticamente entornos de front end y back end y analizando las dependencias de sus archivos de configuración. Este control evaluaba los commits cargados de Git, examinando, para cada uno de ellos, las dependencias del proyecto.

npm



Fig. 2.12. Logo de npm

npm²⁶ (Fig. 2.12) es el administrador de paquetes para JavaScript. Podríamos considerar a npm como una herramienta que permite la colaboración entre los desarrolladores:

²⁴<https://desktop.github.com/>

²⁵<https://tidelift.com/>

²⁶<https://www.npmjs.com/>

- Fomenta el descubrimiento y la reutilización del código de otros equipos de desarrollo.
- Administra el código público y privado con el mismo flujo de trabajo.

En resumen; npm permite instalar, compartir y distribuir código, administrar las dependencias de los proyectos soportados por esta herramienta y recibir feedback de otros usuarios.

Maven



Fig. 2.13. Logo de Maven

Maven²⁷ (Fig. 2.13) es el gestor de dependencias para proyectos Java. Además, actúa como constructor de proyectos generando un archivo de configuración denominado POM.

El POM [8] es un archivo que describe el proyecto en el que se encuentra, incluyendo las dependencias de otros módulos y componentes externos, y el orden de construcción de los elementos. El POM permite introducir distintos perfiles de ejecución (ejecución, test y despliegue).

2.2.3. Entornos de desarrollo

Un entorno de desarrollo es un conjunto de herramientas o programas que conforman una aplicación completa para la programación. Proveen de una interfaz gráfica para el usuario y facilita el proceso integral de la programación.

WebStorm

WebStorm²⁸ (Fig. 2.14) es, considerado por la gran mayoría de desarrolladores, el mejor IDE para desarrollar en JavaScript. Soporta plataformas de desarrollo web como Angular, React o Vue.js, de desarrollo móvil como Ionic, Cordova o React Native y de servidores como Node.js o Meteor.

²⁷<https://maven.apache.org/>

²⁸<https://www.jetbrains.com/webstorm/>



Fig. 2.14. Logo de WebStorm

Spring Tool Suite



Fig. 2.15. Logo de Spring Tool Suite

Spring Tool Suite²⁹ (Fig 2.15) es un entorno de desarrollo basado en Eclipse personalizado para desarrollar aplicaciones Spring. Proporciona un entorno al proyecto Spring que permite su rápida ejecución, aunque también se pueden crear proyectos Java.

2.2.4. Entornos de integración continua

Travis CI

Travis CI³⁰ (Fig 2.16) es un entorno de integración continua que se sincroniza con los repositorios de GitHub y prueba el código en cuestión de minutos. Contiene una grandí-

²⁹<https://spring.io/tools/sts/all/>

³⁰<https://travis-ci.com/>

sima cantidad de plataformas de desarrollo y de bases de datos que permite construir un archivo de configuración ajustado a las necesidades de los desarrolladores.

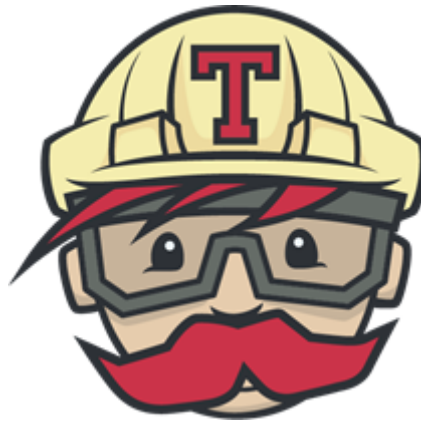


Fig. 2.16. Logo de Travis CI

CircleCI



Fig. 2.17. Logo de CircleCI

CircleCI³¹ (Fig. 2.17) es un entorno en el que se pueden construir entornos personalizados para aplicar flujos de trabajo para controlar la construcción de los proyectos y disfrutar de una asignación de recursos flexible.

Esta herramienta nos fue útil durante toda la fase de desarrollo, menos en la última release (el último mes de desarrollo), ya que, debido a que contuvimos el front y el back end en el mismo repositorio, el archivo de configuración no permitía contemplar esta opción.

³¹<https://circleci.com/>

2.2.5. Bases de Datos

Puesto que tenemos un back end, resulta necesario contar con una base de datos que nos permita almacenar la información introducida a través del front end.

ELK Stack

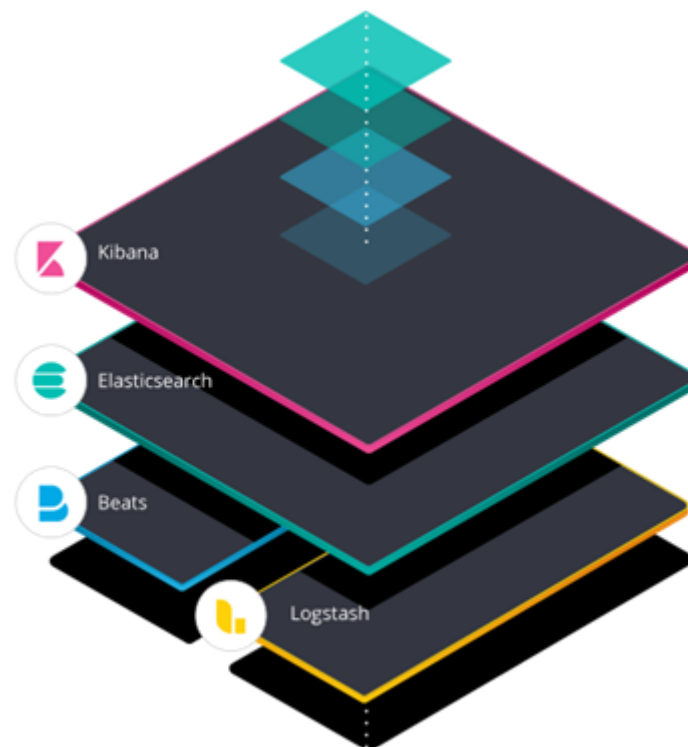


Fig. 2.18. Logo de ELK Stack

ELK Stack³² (Fig. 2.18) es el acrónimo para englobar a los tres proyectos de código abierto más importantes de Elastic: Elasticsearch, Kibana y Logstash. Nosotros hemos utilizado los dos primeros, ya que para la ingesta de datos en Elasticsearch hemos utilizado el cliente back end de Spring.

ELK Stack comienza con Elasticsearch³³, el motor de búsqueda de código abierto, distribuido y basado en JSON. Elasticsearch es fácil de usar, escalable y flexible. Después, la información se ingesta a través de Logstash³⁴ (o el cliente Spring que hemos desarrollado) y se visualiza y analiza con Kibana³⁵.

El hecho de haber seleccionado ElasticSearch como base de datos y no otro tipo como MySQL³⁶ u otras relacionales reside en que las funcionalidades de Elasticsearch están

³²<https://www.elastic.co/elk-stack/>

³³<https://www.elastic.co/products/elasticsearch>

³⁴<https://www.elastic.co/products/logstash>

³⁵<https://www.elastic.co/products/kibana>

³⁶<https://www.mysql.com/>

específicamente diseñadas para el almacenamiento y recuperación de información no estructurada como los logs.

2.2.6. Despliegue

El despliegue de un proyecto software comienza cuando ha sido suficientemente probado, ha sido aceptado para su liberación y ha sido distribuido en el entorno de producción.

Docker

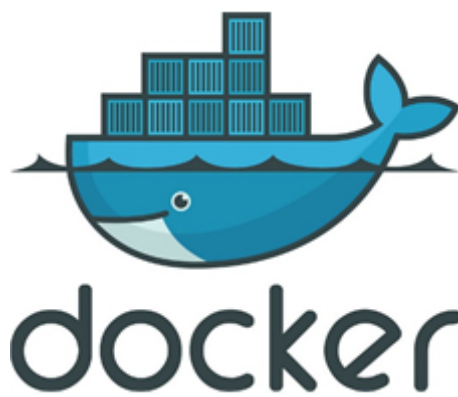


Fig. 2.19. Logo de Docker

Docker³⁷ (Fig. 2.19) es una plataforma de despliegue que aborda cada aplicación a través de la nube híbrida. Docker automatiza el despliegue de aplicaciones dentro de contenedores de software.

Un contenedor [9] es un paquete de elementos que permiten ejecutar una aplicación determinada en cualquier sistema operativo.

De manera similar a como hace una máquina virtual con el hardware del servidor, los contenedores virtualizan el sistema operativo de un servidor. Docker se instala en un servidor y proporciona comandos sencillos que el usuario puede utilizar para crear, iniciar o detener contenedores.

Uno de los motivos más importantes por los que utilizar Docker es que permite entregar código de una forma rápida (y, por tanto, transferirlo muy sencillamente).

³⁷<https://www.docker.com/>

Docker Compose

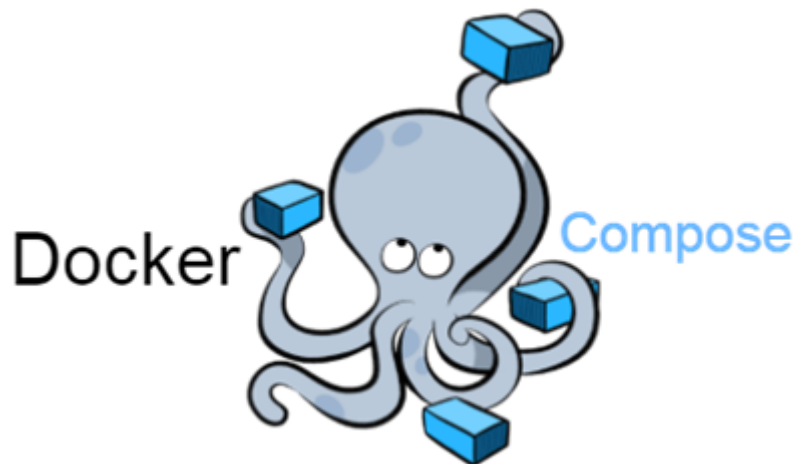


Fig. 2.20. Logo de Docker Compose

Docker Compose³⁸ (Fig. 2.20) es una herramienta para definir aplicaciones de Docker que contienen múltiples contenedores, estableciendo una relación bidireccional entre los mismos.

Docker Compose basa su funcionamiento en la orquestación. La orquestación es la planificación de contenedores, administración en conjunto y la posibilidad de provisionar hosts adicionales.

Con Docker Compose hemos podido orquestar los contenedores del front end, back end y del ELK Stack.

2.2.7. Otras herramientas

Postman

Postman³⁹ (Fig. 2.21) es una herramienta para probar aplicaciones con servicios REST elaboradas por otros o por uno mismo. Ofrece una interfaz muy sencilla lista para realizar peticiones HTTP sin la necesidad de escribir tests para la API REST (hemos utilizado Postman como otra herramienta de testing para el back end).

Postman además permite programar peticiones para ser lanzadas cuando nosotros le indiquemos, además de la opción de hacer un debug en las peticiones y de almacenarlas en colecciones, para ser fácilmente transportadas.

³⁸<https://docs.docker.com/compose/>

³⁹<https://www.getpostman.com/>



Fig. 2.21. Logo de Postman

Postman nos permitió, como medida extra de testing, probar exhaustivamente la API REST en busca de fallos e incongruencias. Se probaron todos los métodos disponibles en los controladores, así como realizar peticiones de pruebas a Elasticsearch en las fases tempranas del desarrollo de TestLogAnalyzer a modo de asegurar que los datos estaban siendo ingestados correctamente en el ELK Stack.

Compodoc



Fig. 2.22. Logo de Compodoc

Compodoc⁴⁰ (Fig. 2.22) es una herramienta que puede ser instalada por npm y que genera, de forma automática, la documentación de una aplicación Angular.

Es de código abierto y se instala localmente. También posee un motor de búsqueda y exporta la información en un formato HTML ejecutable y transportable (de forma semejante a la Javadoc).

⁴⁰<https://compodoc.github.io/compodoc/>

2.3. Metodología

La metodología de desarrollo que se ha llevado a cabo a lo largo de estos diez meses de desarrollo ha sido iterativa e incremental.

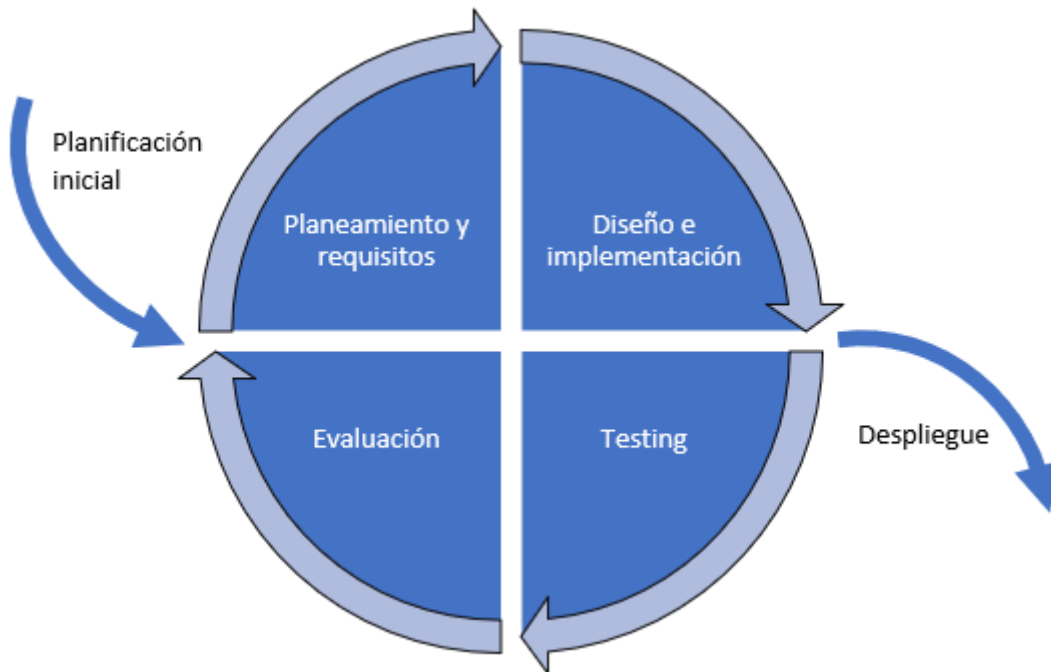


Fig. 2.23. Esquema del desarrollo iterativo e incremental

En el desarrollo iterativo [10] (Fig. 2.23) el proyecto se planifica en diversos bloques temporales llamados iteraciones. En cada iteración, el producto evoluciona, a partir de los resultados completados en las interacciones anteriores, añadiendo nuevos objetivos o requisitos o mejorando los que ya fueron completados con anterioridad. Un aspecto fundamental para guiar el desarrollo iterativo e incremental es la priorización de los objetivos/requisitos en función del valor que aportan al cliente.

Este ciclo es una de las bases de un proyecto ágil [11], que acerca al cliente a un entorno de trabajo cercano al equipo de desarrollo. Las iteraciones duraron entre 2 y 4 semanas, según las buenas prácticas de esta metodología de desarrollo.

En el cuarto capítulo profundizaremos en cómo se llevó a cabo este proceso.

3. MANUAL DE USO

Con el objetivo de que el lector pueda entender mejor las funcionalidades ofrecidas por la herramienta TestLogAnalyzer desarrollada en este trabajo, se ha decidido incluir un breve Manual de Uso. En el siguiente capítulo se explicará en detalle cómo ha sido implementada esta herramienta.

3.1. Visualización de los proyectos

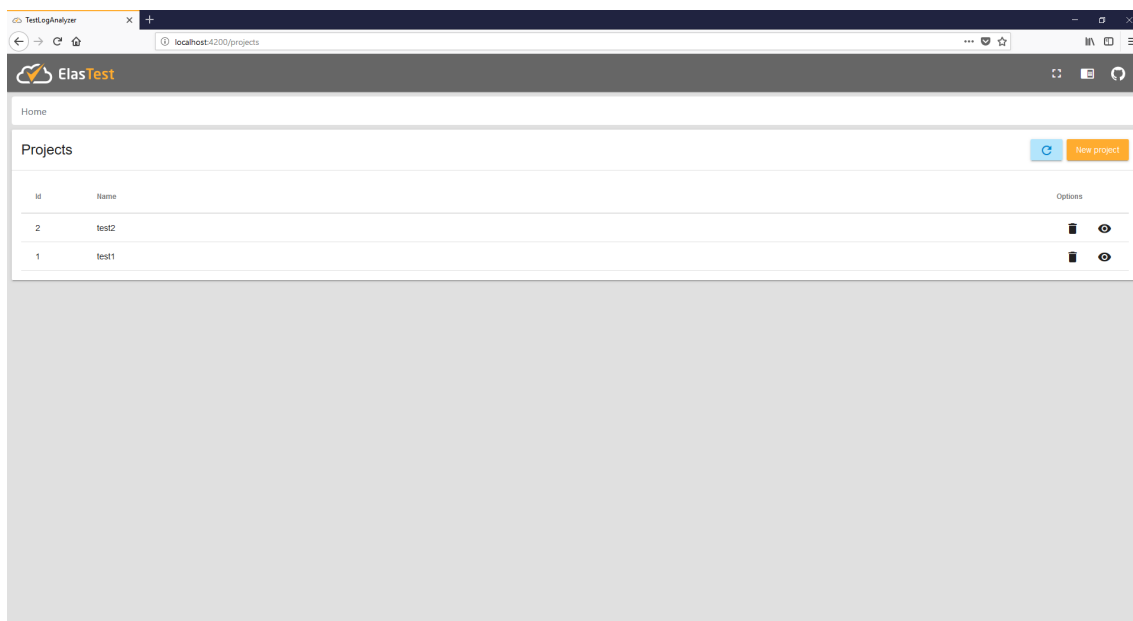


Fig. 3.1. Página principal de la sexta versión del TestLogAnalyzer

La página principal (Fig. 3.1) representa el primer contacto del usuario con la aplicación. En esta página, el usuario podrá consultar los proyectos creados con anterioridad, acceder a ellos y/o eliminarlos.

Como se definió anteriormente, un proyecto no es más que un conjunto de ejecuciones de tests que forman parte de un sistema.

3.2. Creación de proyectos y subida de logs

Los proyectos se podrán crear de dos formas distintas:

1. Mediante la interfaz gráfica. TestLogAnalyzer ofrece una interfaz gráfica dedicada para el usuario, cuya utilidad es la subida de ejecuciones para la creación y/o ampliación de los proyectos.

- Mediante una petición a la API REST. El back end ofrece una API REST que contiene peticiones para crear, eliminar o visualizar proyectos, ejecuciones y tests. Adicionalmente, la API REST, en cuanto a la creación de proyectos y ejecuciones, reconoce el comando cURL para tal acción, de tal manera que el usuario puede, directamente, desde sus entornos de integración continua, almacenar la información en TestLogAnalyzer tras la finalización de la ejecución de los tests. El comando cURL es:

```
1 curl -F test1.txt -F test2.txt ... -F testN.txt -F surefire1.xml -F
   surefire2.xml ... -F surefireN.xml http://localhost:8443/api/
   projects/{project}
```

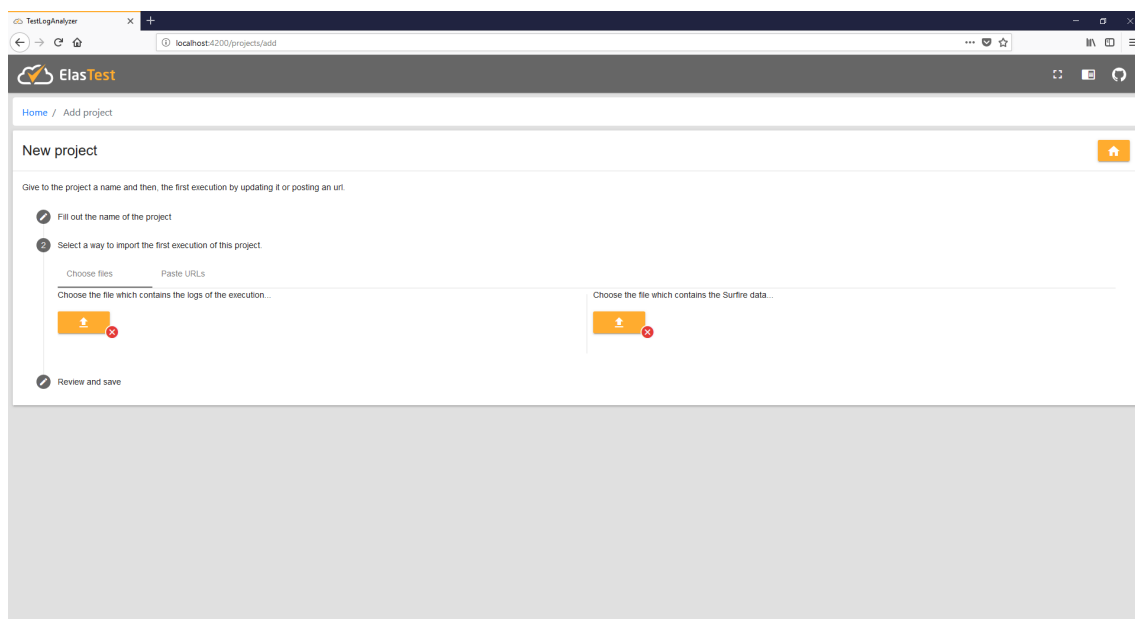


Fig. 3.2. Página de creación de la sexta versión del TestLogAnalyzer

La interfaz gráfica (Fig. 3.2) permite la adición de ejecuciones, y le permitirá al usuario introducir un nombre al proyecto según se encuentre en la página de creación de proyectos o no: si el proyecto ya ha sido creado y el usuario desea introducir nuevas ejecuciones, entonces no debería tener la opción de añadir un título al proyecto, ya que, accediendo a él, es como puede introducir nuevas ejecuciones. En caso contrario, debe asignar un nombre al proyecto, y la interfaz gráfica no le permitirá subir la información hasta que no lo haya hecho.

Una ejecución está compuesta por un documento de texto (.txt) que contiene los logs del entorno en el que se han ejecutado los tests siguiendo el formato:

```
« % {yyyy-MM-dd hh:mm:ss.SSS} %5level — [ %t] &logger{36} : %m %n »
```

```
«2017-07-12 10:51:33.263 INFO — [ main] c.f.backend.e2e.FullTeachingTestE2E :
Adding test course»
```

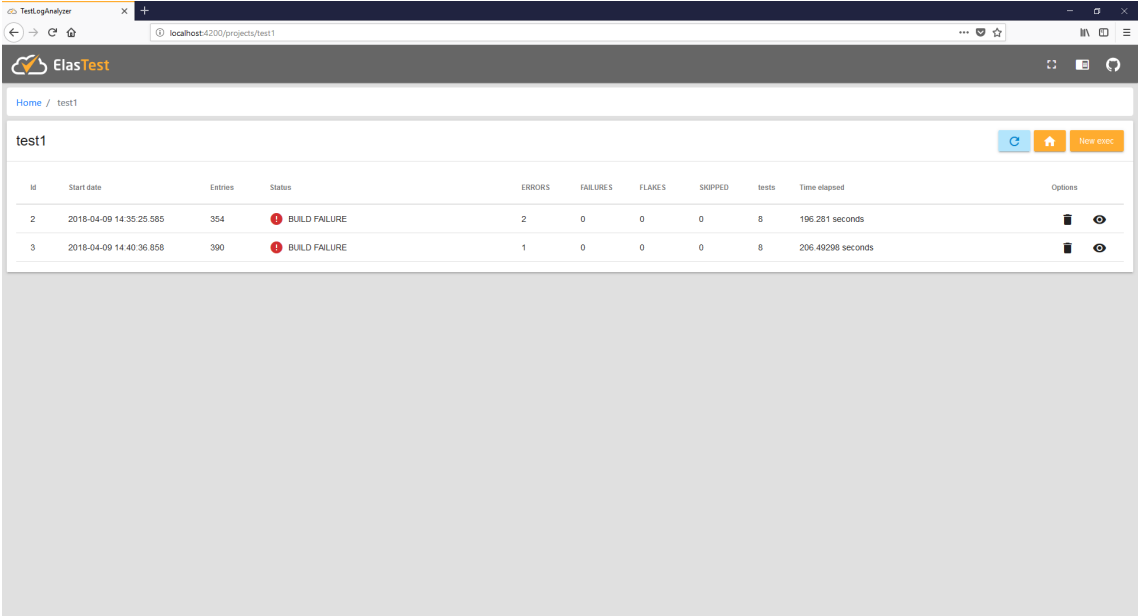
Para que el parser interprete correctamente qué logs forman parte de un test, es necesario que el desarrollador establezca una marca de inicio y una de fin en cada test para tal fin. La marca de inicio (o final) será un log que seguirá la estructura de cualquier log pero incluyendo como mensaje (por ejemplo):

```
«2017-07-12 10:51:33.263 INFO — [ main] c.f.backend.e2e.FullTeachingTestE2E :
#### Start test: oneToOneChatInSessionChrome»
```

Además, una ejecución está compuesta adicionalmente por un documento XML (.xml) elaborado con el Surefire Report Plugin de Maven⁴¹, que resume la información de cada ejecución con un formato estructurado y de fácil interpretación, que acompaña a los logs.

El usuario debe, por tanto, facilitar estos dos documentos para cada clase de test, es decir, si una ejecución contiene tres clases a probar, se subirán un total de tres documentos de texto y tres documentos XML (aunque también funciona con un único fichero de texto que tenga todas las ejecuciones). Los ficheros de texto deberán subirse en orden de creación (los más antiguos primero), para favorecer un orden lógico. Esto habría que tenerlo en cuenta si se suben a mano, ya que, si se utiliza un entorno de integración continua para subirlos, éste lo hará automáticamente. En apéndices hay disponible un ejemplo de cada tipo de fichero.

3.3. Visualización de un proyecto



The screenshot shows a web browser window with the address bar displaying 'localhost:4200/projects/test1'. The page header includes the 'ElasticTest' logo and navigation links. The main content area is titled 'test1' and contains a table with the following data:





Id	Start date	Entries	Status	ERRORS	FAILURES	FLAKES	SKIPPED	tests	Time elapsed	Options
2	2018-04-09 14:35:25.585	354	❌ BUILD FAILURE	2	0	0	0	8	196.281 seconds	 
3	2018-04-09 14:40:36.858	390	❌ BUILD FAILURE	1	0	0	0	8	206.49298 seconds	 

Fig. 3.3. Página de visualización de proyecto de la sexta versión del TestLogAnalyzer

En la página principal, cuando el usuario selecciona la opción de visualizar proyecto, accede a la vista mostrada en Fig. 3.3. En ella, el usuario podrá visualizar las ejecuciones

⁴¹<http://maven.apache.org/surefire/maven-surefire-report-plugin/>

que pertenecen a ese proyecto, eliminarlas y/o añadir nuevas.

Además, gracias al Surefire Report, cada ejecución viene acompañada de un conjunto de detalles que son leídos de estos archivos en su subida, sin la necesidad de realizar ningún cálculo computacional derivado de los documentos de texto que contienen los logs. Los detalles que se muestran son:

- Número de entradas (entries). Representa la cantidad de las líneas de la ejecución, tanto las añadidas por Maven como las que contienen la información sobre los tests. Las excepciones causadas por errores en los tests se corresponden con una única línea.
- Estado de la ejecución (status). El estado de una ejecución será exitoso si y solo si todas las ejecuciones se han resuelto sin errores.
- Número de errores (errors), de fallos (failures), de tests imprevisibles (flaky tests), de test ignorados (skipped tests) y de tests normales (tests). Estos datos son de mucha utilidad, ya que se puede apreciar, sin realizar ningún tipo de comparación, la evolución entre varias ejecuciones de un mismo proyecto.
- Tiempo requerido para la ejecución de todos los tests (time elapsed).

La asignación del número identificador de las ejecuciones es incremental y no se reutiliza una vez la ejecución ha sido eliminada.

3.4. Visualización y comparación de una ejecución

Cuando el usuario accede a la página, ésta se abre en modo visualización, aunque, si ha habido ejecuciones anteriores a la seleccionada, será seleccionada la anterior por defecto. La visualización tiene cuatro modos:

- Visualización completa de logs (complete logs). Como en Fig. 3.4, se muestran todos los logs de la ejecución, incluyendo los generados por Maven. En este modo se visualiza toda la ejecución completa, desde la primera línea hasta la última. Ésto permite al usuario de la aplicación visualizar de forma completa la ejecución.
- Visualización sin mensajes de Maven (only tests logs). Como en Fig. 3.5, se mostrarán únicamente los logs asociados a ejecuciones de tests, eliminando los mensajes de Maven. En este modo de visualización, se eliminan las líneas que no aportan valor alguno, ya que son generadas por Maven e independiente de la ejecución de los tests. A efectos prácticos, sólo se mostrarían las líneas de cada test.

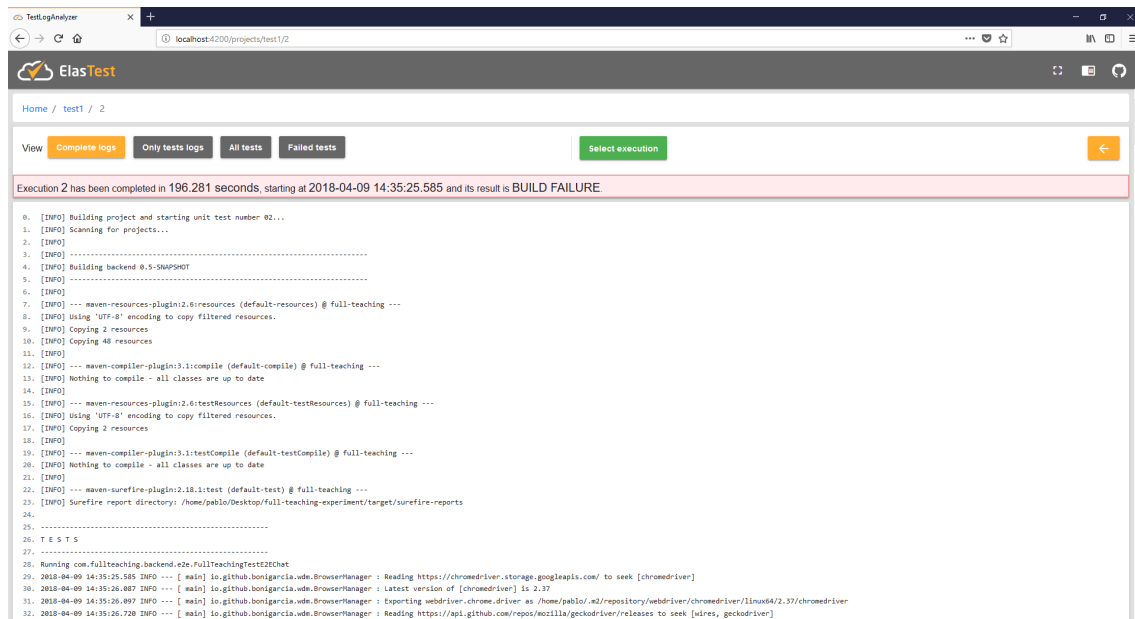


Fig. 3.4. Modo de visualización completa

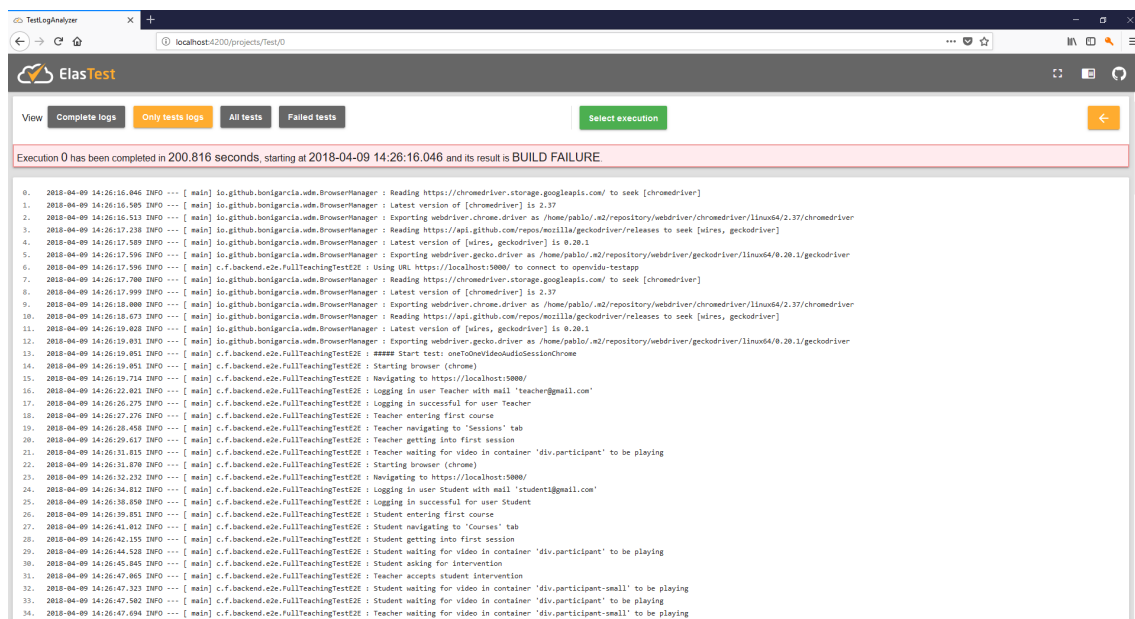


Fig. 3.5. Modo de visualización sin mensajes de Maven

- Visualización de los tests (all tests). Como en Fig. 3.6, se mostrarán los logs asociados a cada test. Estos logs se organizarán por test, en orden de aparición en la ejecución. A su vez, los tests estarán organizados por clase.

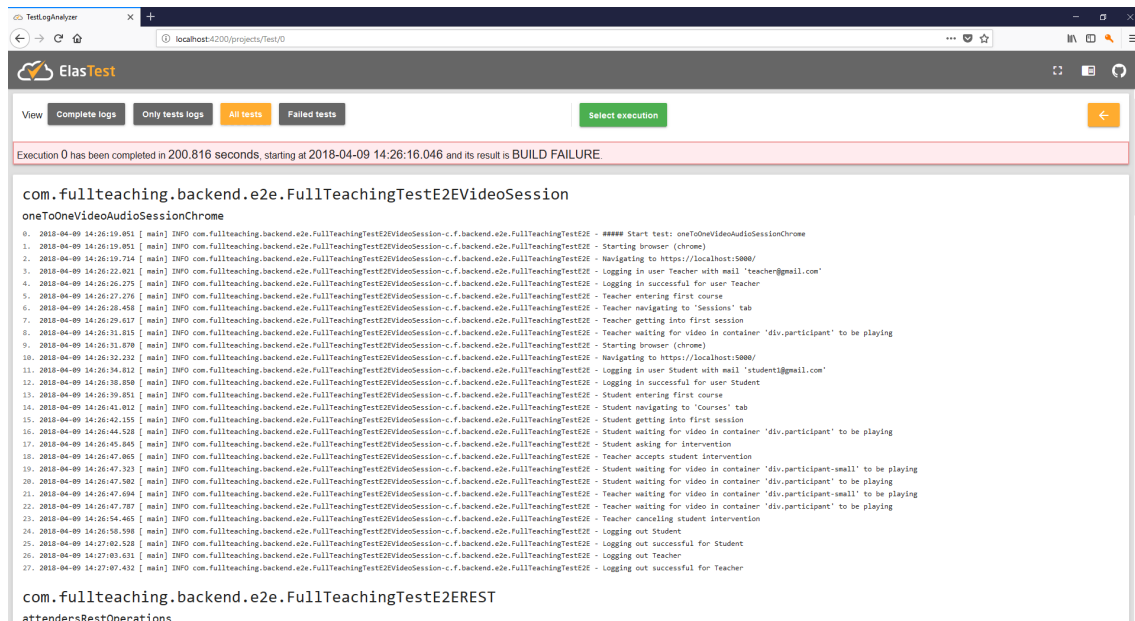


Fig. 3.6. Modo de visualización de los tests

- Visualización de los tests fallidos (failed tests). Como en Fig. 3.7, solo sólo se mostrarán los logs de los tests fallidos, organizados según el test al que pertenezcan. Estos, a su vez, también estarán organizados por clase.

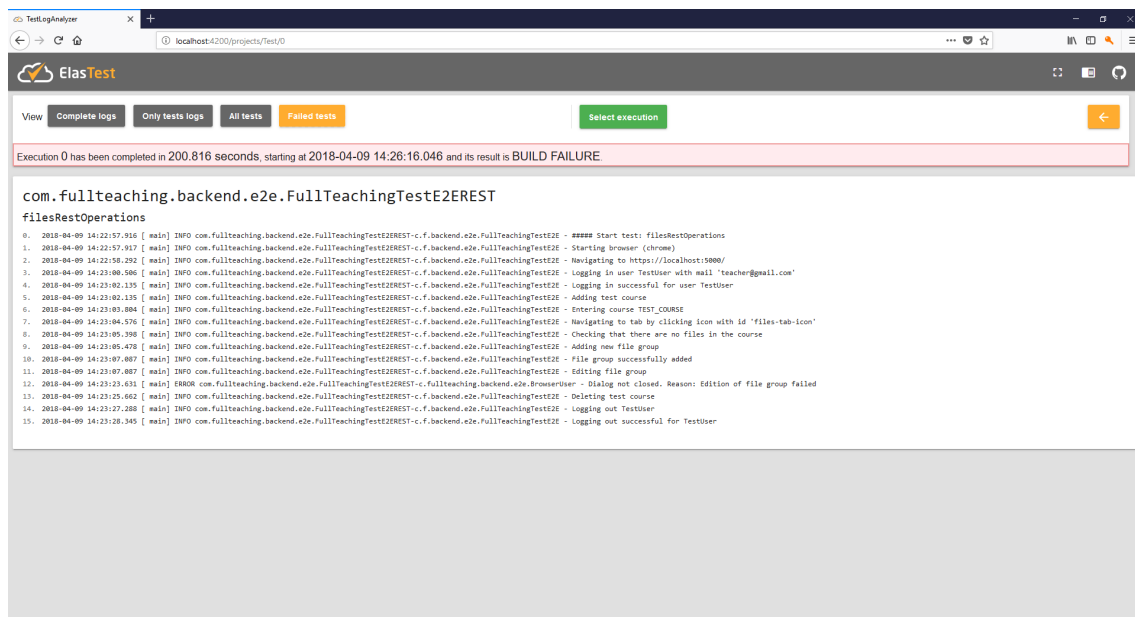


Fig. 3.7. Modo de visualización de los tests fallidos

La comparación, sin embargo, tiene tres modos:

- Comparación completa (complete). Tal y como se muestra en Fig. 3.8 Cada línea se comparará tal y como se ha registrado. No se realiza ningún tratamiento de la misma como la eliminación de la fecha y hora de generación.

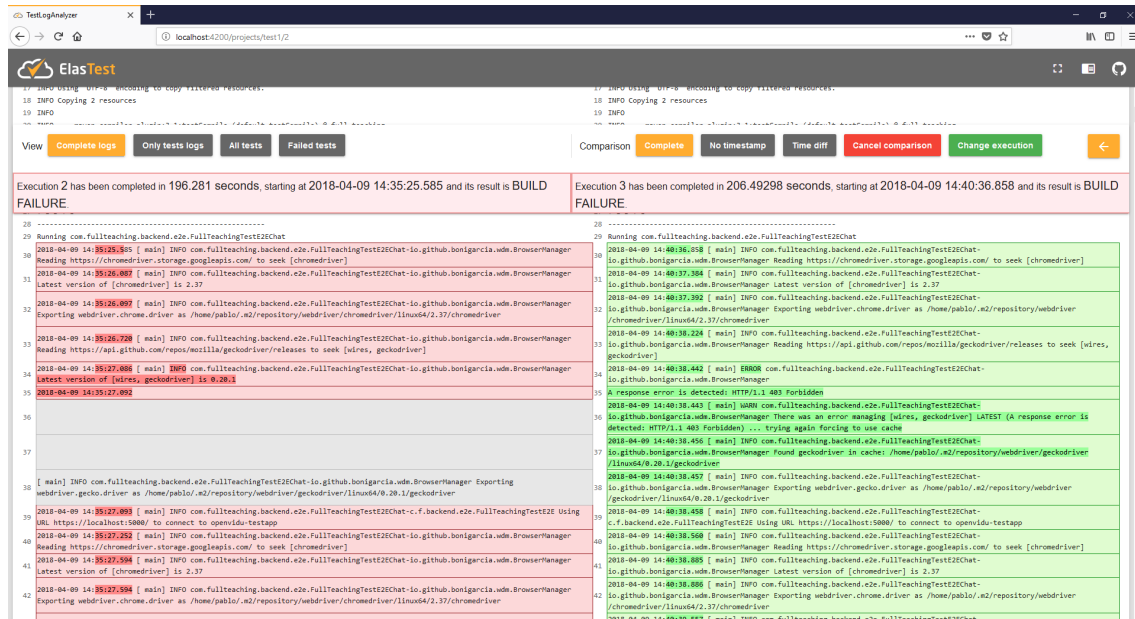


Fig. 3.8. Modo de comparación completa con visualización completa

- Comparación sin timestamp (no timestamp). Mostrado en Fig. 3.9, los logs se compararán ocultando el campo del timestamp, el cual será sustituido por un carácter vacío.

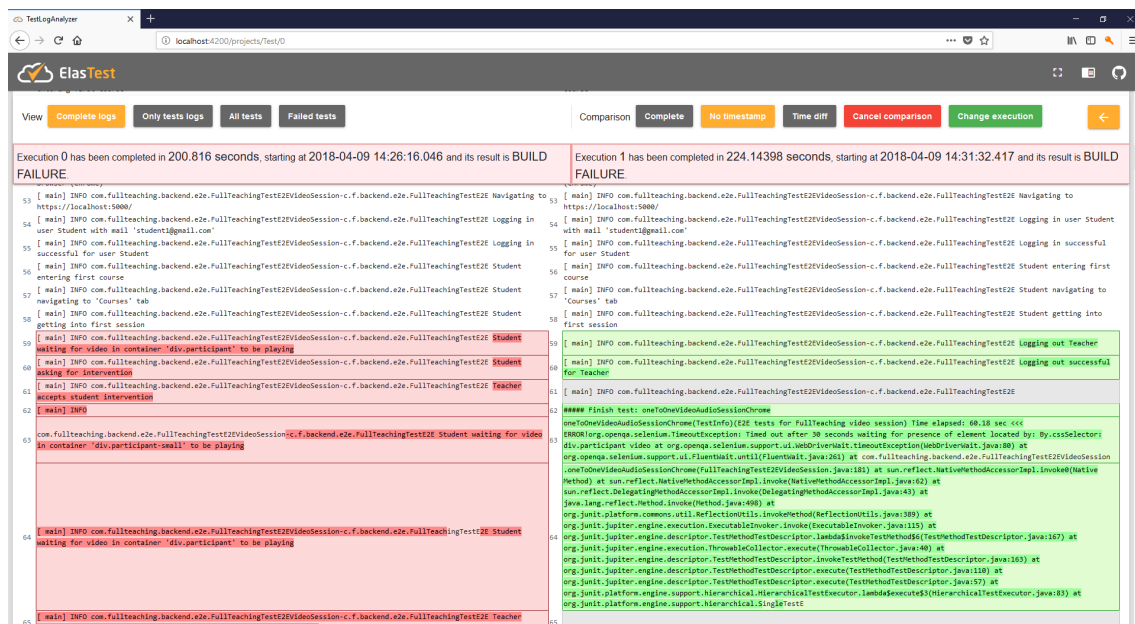


Fig. 3.9. Modo de comparación sin timestamp con visualización completa

- Comparación con diferenciación temporal (time diff). Como se muestra en Fig. 3.10, este modo de comparación es útil para saber cuánto tiempo transcurre entre una línea de log y la siguiente. El objetivo de esta transformación es medir el tiempo relativo (en ms) entre los logs, ya que puede ayudar a detectar problemas de ejecuciones más lentas.

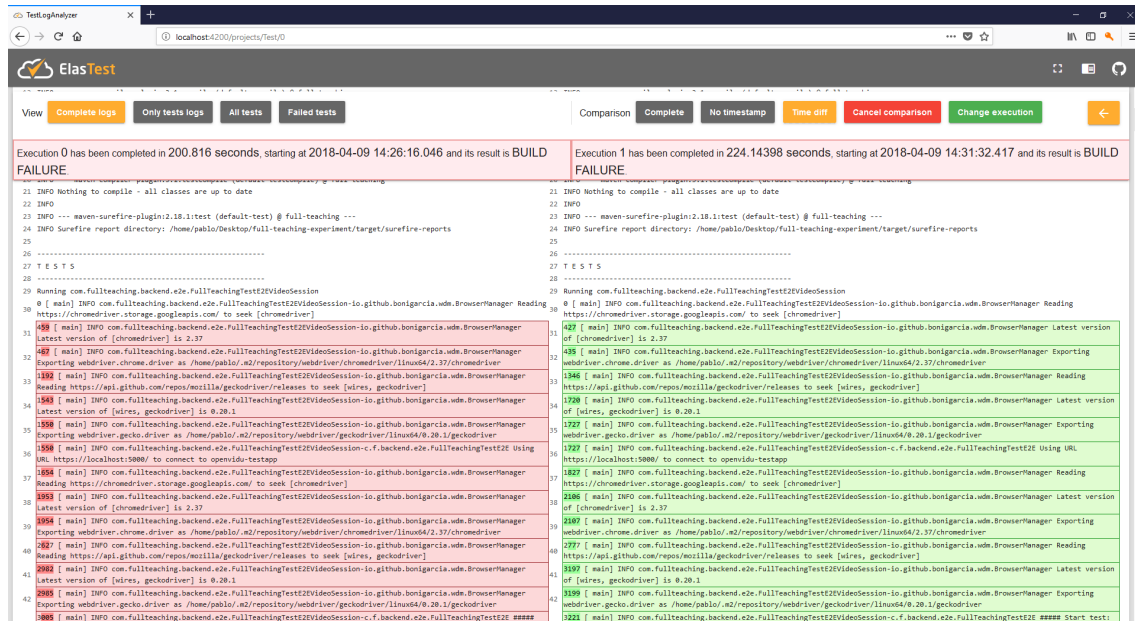


Fig. 3.10. Modo de comparación con diferenciación temporal y visualización completa

4. DESCRIPCIÓN INFORMÁTICA

4.1. Desarrollo iterativo e incremental

A través de numerosas reuniones a lo largo de los meses de desarrollo, el producto ha ido evolucionando progresivamente, diferenciándose tres posibles fases (hitos):

1. **Fase 1: Almacenamiento y visualización de la información.** Establecimiento de la tecnología de almacenamiento de los logs y su primera visualización. En esta fase se publicaron las releases beta, 1.0.0 y 2.0.0 (primera y segunda versión, respectivamente).
2. **Fase 2: Análisis de la información.** Creación de métodos primitivos de comparación con el objetivo de probar y modificar el algoritmo de diferenciación. En esta fase se publicó la release 3.0.0 (tercera versión).
3. **Fase 3: Producto autónomo.** Una vez se han establecido los mecanismos de almacenamiento, visualización y comparación, se define una interfaz acorde con Elast-Test, se proporcionan mecanismos de visualización y comparación adecuados y se ofrecen los mecanismos para utilizar la aplicación desde un entorno de integración continua. En esta fase se publicaron las releases 4.0.0, 5.0.0 y 6.0.0.

4.1.1. Fase 1: Almacenamiento y visualización de la información

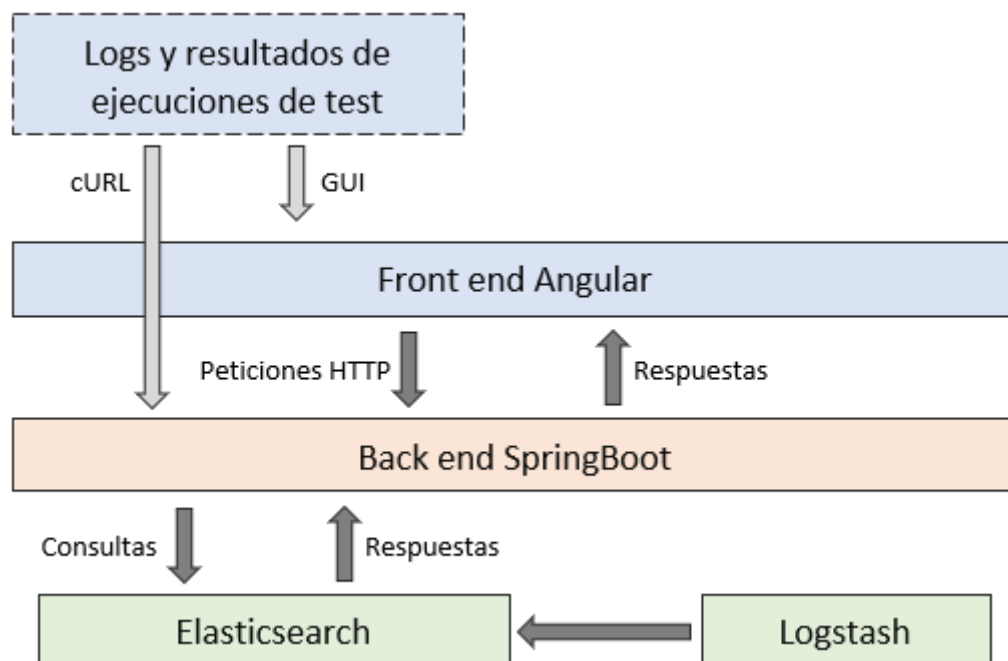


Fig. 4.1. Diagrama arquitectónico de TestLogAnalyzer

Fig. 4.1 muestra la estructura de TestLogAnalyzer. Está conformado, como se ha mencionado, por una base de datos, Elasticsearch, encargada del almacenamiento de los logs, los cuáles ingesta por medio de consultas del back end o mediante Logstash. El back end contiene la API REST que atiende peticiones por parte del cliente, que está dotado de una interfaz gráfica de usuario que facilita al usuario la subida de ejecuciones al sistema. La API también permite la subida de ejecuciones utilizando cURL. La versión beta (Fig 4.2) se centró en la búsqueda de mecanismos para la subida y almacenamiento de la información, recurriendo al ELK Stack para tal fin.

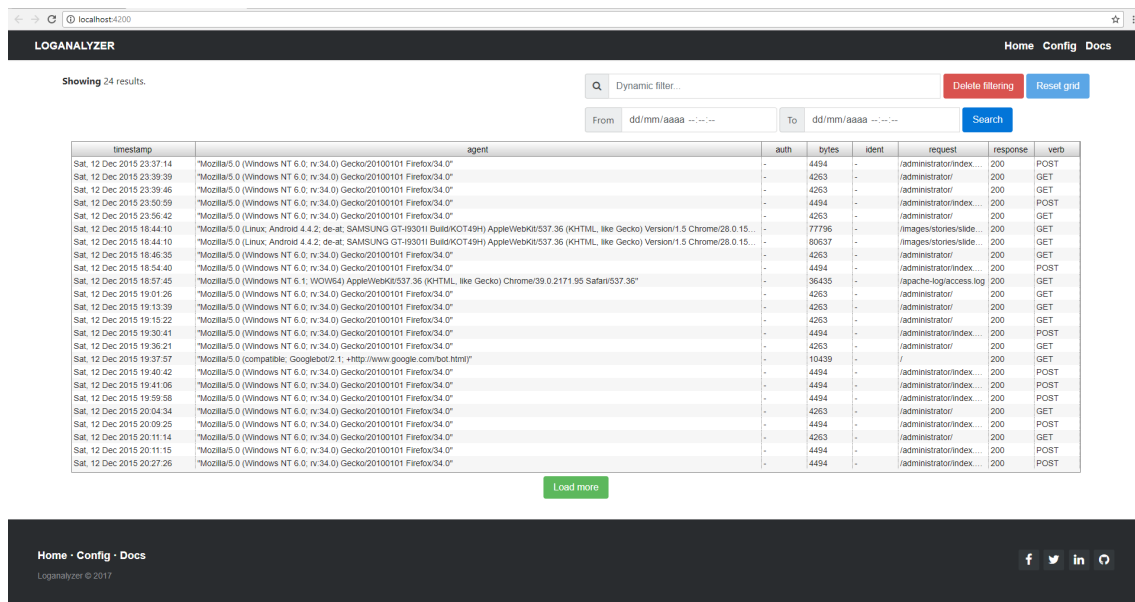


Fig. 4.2. Versión beta del TestLogAnalyzer

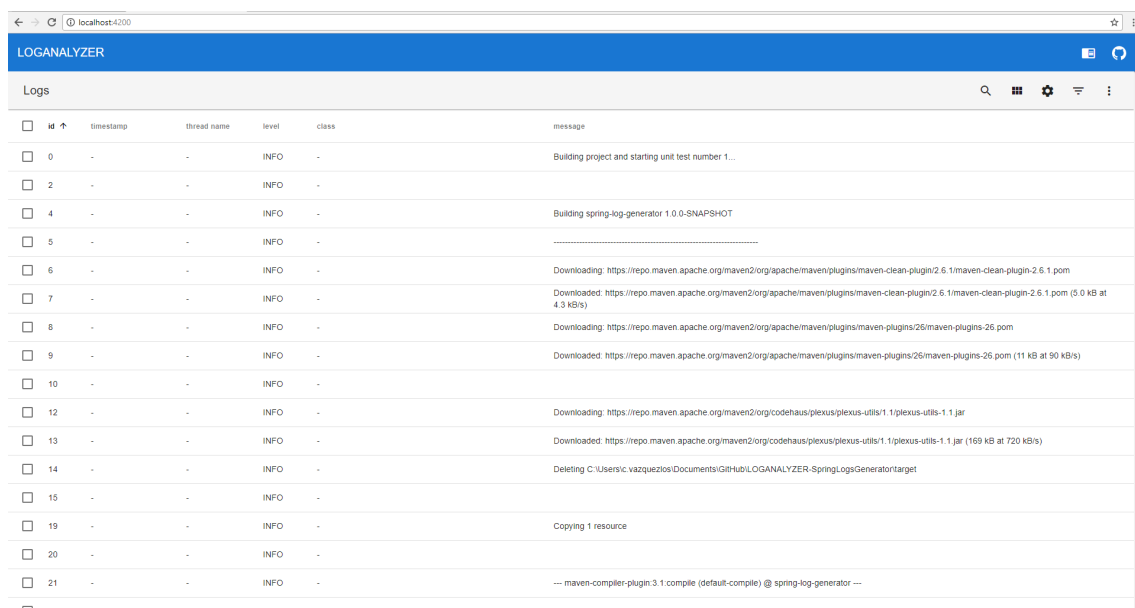


Fig. 4.3. Primera versión del TestLogAnalyzer

La primera versión (Fig 4.3) del proyecto consistió en aplicar librerías de estilos y

refinar el diseño anterior, realizando un proyecto responsive para que funcionase con independencia del medio electrónico en el que se ejecutase.

Se utilizó como librería de estilos Teradata Covalent y se definió el formato de entrada de los logs a Logstash. También, fue añadido un apartado en el que se realizarían las comparaciones, de cara a futuras versiones.

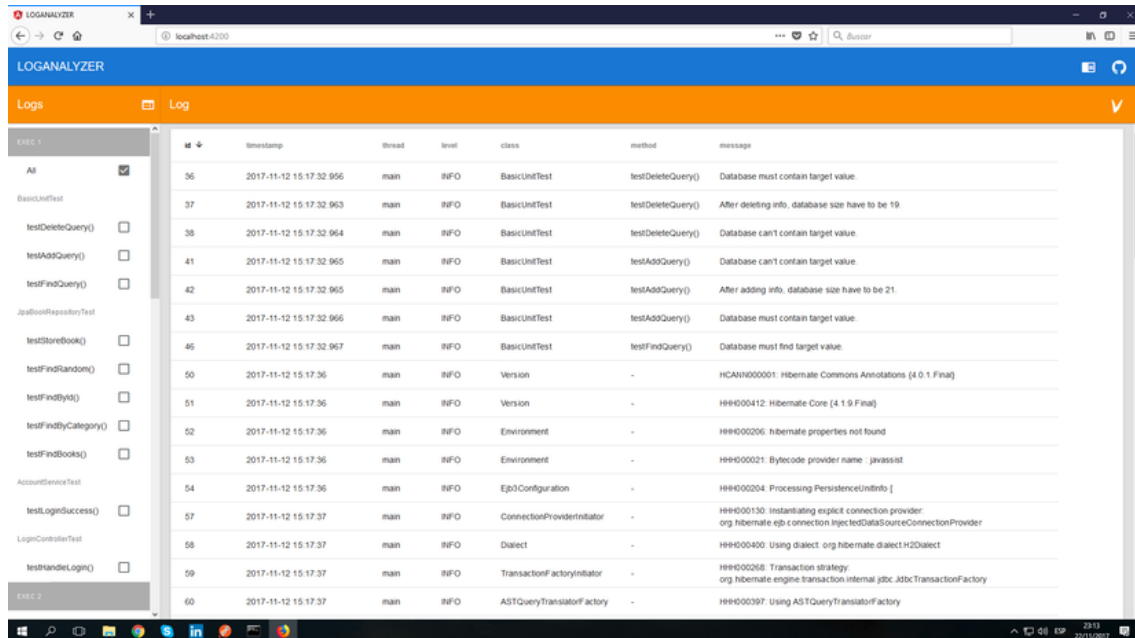


Fig. 4.4. Página principal de la segunda versión del TestLogAnalyzer

Fig. 4.4 muestra la segunda versión, implementada en su totalidad con Teradata Covalent.

En ella, se introdujeron cambios sustanciales y que añadían cierta lógica computacional. En primer lugar, el concepto de ejecución en términos de test. Podríamos considerar una ejecución como el almacenamiento de una ristra de logs en un documento de texto para su posterior almacenamiento en Elasticsearch.

Las ejecuciones contendrán los propios logs, organizados por tests (métodos que se ponen a prueba y los cuáles devuelven un conjunto de logs informativos), junto con el estado de cada uno de ellos y el resultado final de la ejecución. También, mensajes de estado de Maven.

En esta versión, el usuario solo podía añadir ejecuciones ingestándolas a través del ELK Stack (utilizando Logstash) mediante la consola de comandos, por lo que no disponía de una interfaz propia para ello.

Además, se podrían realizar comparaciones muy primitivas y sencillas, señalando, para ello, el test y la ejecución.

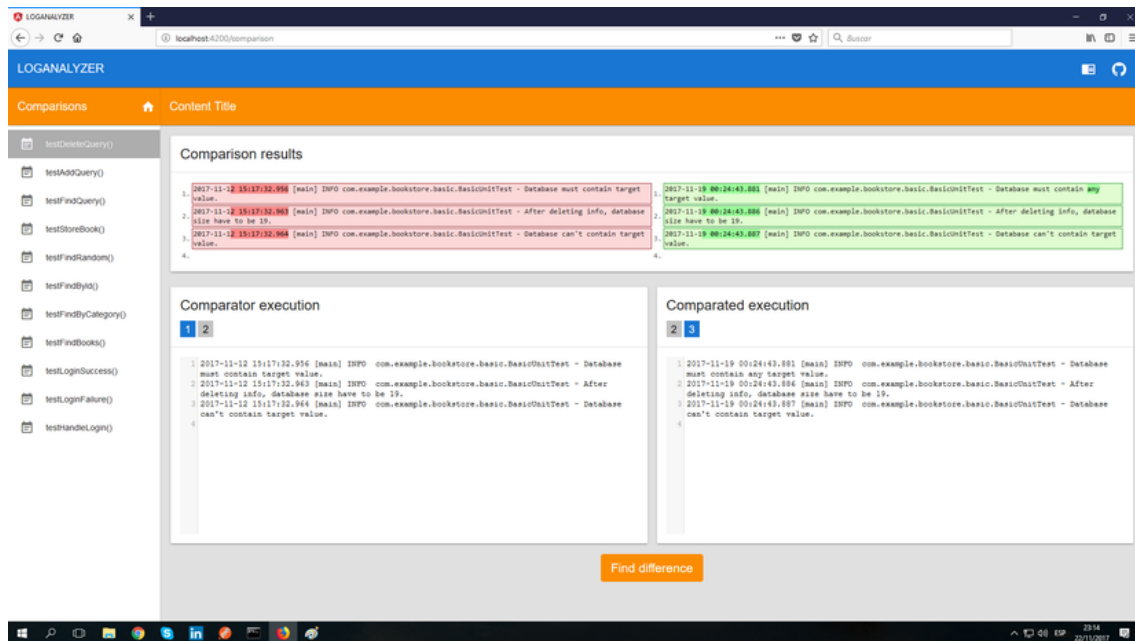


Fig. 4.5. Página de comparaciones de la segunda versión del TestLogAnalyzer

La comparación (Fig. 4.5) no permitía que el usuario pudiese elegir diferentes modos para ignorar, por ejemplo, el ruido de la fecha y poder, de esa forma, comparar de una forma más precisa.

4.1.2. Fase 2: Análisis de la información

Los distintos modos de ejecución llegarían en la tercera versión de la aplicación: normal, sin timestamp y con diferenciación temporal, que tiene en cuenta la diferencia en milisegundos entre los logs.

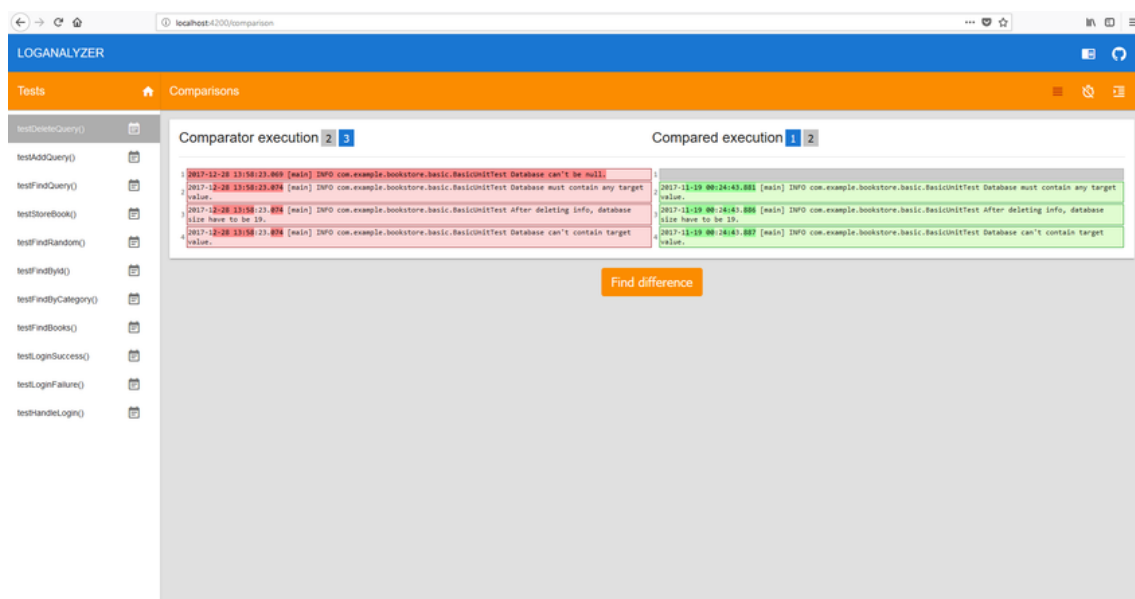


Fig. 4.6. Página de comparaciones de la tercera versión del TestLogAnalyzer

En esta tercera versión, se conservó el diseño de la página principal y únicamente se modificó el diseño y la estructura de la página de comparaciones (Fig. 4.6), facilitando la utilización de la interfaz y eliminando la previsualización del test antes de realizar la comparación, ya que consideramos que era innecesaria.

En este punto, también fue desarrollado un cliente para Elasticsearch en Spring, que permitía la rápida subida de archivos, pero todavía no contaba con la API REST, por lo que solo era posible mediante la consola de comandos, ejecutando el cliente en el mismo directorio que la ejecución.

En este estado del desarrollo, el proyecto era lo que podríamos esperar y cumplía el objetivo principal de la aplicación, que era analizar la información, compararla y visualizar esa comparación para poder ser examinada por el usuario.

4.1.3. Fase 3: Producto autónomo

Tras tener una idea sobre los siguientes pasos a seguir, el siguiente objetivo era trabajar tanto en el diseño de la aplicación para que pudiese ser utilizada como módulo de ElasTest (incluyendo su código de colores, su logo. . .), como en el aspecto funcional (modificando la estructura de las ejecuciones para un mayor aprovechamiento del potencial de las comparaciones).

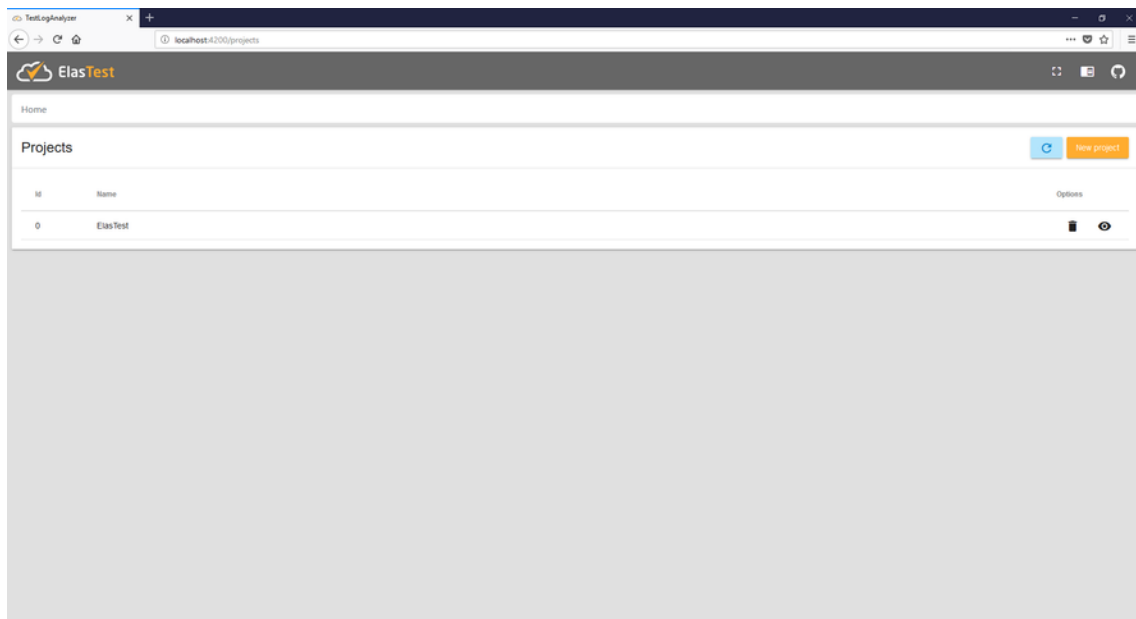


Fig. 4.7. Página principal de la cuarta versión del TestLogAnalyzer

La cuarta versión (Fig. 4.7) cumplía las expectativas en cuanto a lo que a la visualización se refiere, ya que consistía en un diseño maduro, responsive y con una interfaz limpia y amigable. En cuanto a la funcionalidad, la página principal albergaba los proyectos (un conjunto de ejecuciones), y, éstos, se podían crear o eliminar utilizando el GUI.

Cada proyecto contenía un conjunto de ejecuciones y, cada una de ellas, estaba estructurada según el componente que testease, en un simple sistema de pestañas. Además, cada ejecución era accesible y permitía una rápida visualización de los logs que la componían.

Ahora se podían comparar ejecuciones completas (y no solo tests), lo que facilitaba en gran medida la labor del usuario de la aplicación, evitando que tuviera que recorrer todos los tests de una ejecución y, para cada uno de ellos, seleccionar la ejecución concreta para comparar (lo cual era engorroso para un gran número de tests).

La quinta versión incluía modos de visualización y comparación simultáneos, de tal manera que se pudiesen combinar para unos resultados más personalizados según las distintas circunstancias.

Se mantuvieron el resto de las páginas intactas, y se modificó la página de comparaciones para ofrecer criterios de visualización y comparación, además de un novedoso contexto para que el usuario supiese en todo momento el estado de la ejecución (si había fallado o no).

La última versión, la sexta, es la última fase de la aplicación y supone la culminación de este proyecto. El contenido de la sexta versión será explicado a continuación.

4.2. Requisitos

Debido a que las iteraciones del desarrollo de TestLogAnalyzer incluían un incremento en la complejidad funcional de la herramienta, los requisitos asociados con la última iteración se muestran en Tab. 4.1.

Id.	Nombre	Resumen
0	Almacenamiento de información en Elasticsearch	Permitirá subir los logs (proyectos y ejecuciones) a través de diversos medios a Elasticsearch.
1	Visualización de la información	Se podrán visualizar proyectos, ejecuciones y logs (visualización básica).
2	Modos de visualización de ejecuciones	Una ejecución deberá ofrecer distintos modos de visualización.
3	Modos de comparación de ejecuciones	Una ejecución deberá ofrecer distintos modos de comparación.
4	Mezcla de modos	La aplicación deberá soportar todas las combinaciones entre modos de visualización y de comparación.
5	Eliminación de información en Elasticsearch	Permitirá eliminar la información alojada en Elasticsearch libremente.

6	Tiempos de ejecución reducidos	Los usuarios no deberán esperar más de lo estrictamente necesario al realizar alguna función para la que la aplicación esté lista.
7	Estructura y diseño según ElasTest	La aplicación deberá seguir los patrones establecidos por ElasTest completamente.
8	Creación de una API REST	TestLogAnalyzer ofrecerá una API REST para permitir realizar de forma automática la subida, actualización y eliminación de información.
9	Empaquetado de la aplicación	Cada uno de los componentes de TestLogAnalyzer deberán ser empaquetados con Docker.
10	Orquestación y ejecución con comando simple	Los componentes se deberán orquestar para lanzar la aplicación usando un único comando.
11	Integración continua	La aplicación deberá contener mecanismos de integración continua que aseguren la calidad del desarrollo con cada commit.
12	Calidad de codificación	La aplicación deberá hacer uso de alguna herramienta que revise la calidad del código mientras evoluciona.

Tab. 4.1. Requisitos de la última iteración

4.3. Arquitectura

El proyecto contiene un front end, un back end y una base de datos, que operan en armonía para ofrecer la funcionalidad recientemente mencionada. Además, la posibilidad de empaquetar la aplicación a través de Docker la convierte en transportable y autónoma, capaz de ser añadida en, prácticamente, cualquier sistema.

Anteriormente, en Fig. 4.1, se mostró la arquitectura de TestLogAnalyzer. En los próximos apartados examinaremos a fondo el front end y back end.

4.3.1. Front end

El front end (Fig. 4.8) está compuesto por un componente para cada una de las páginas, sin contar los componentes PublicComponent, ProjectComponent y ExecComponent, que son componentes para la utilización correcta de las direcciones.

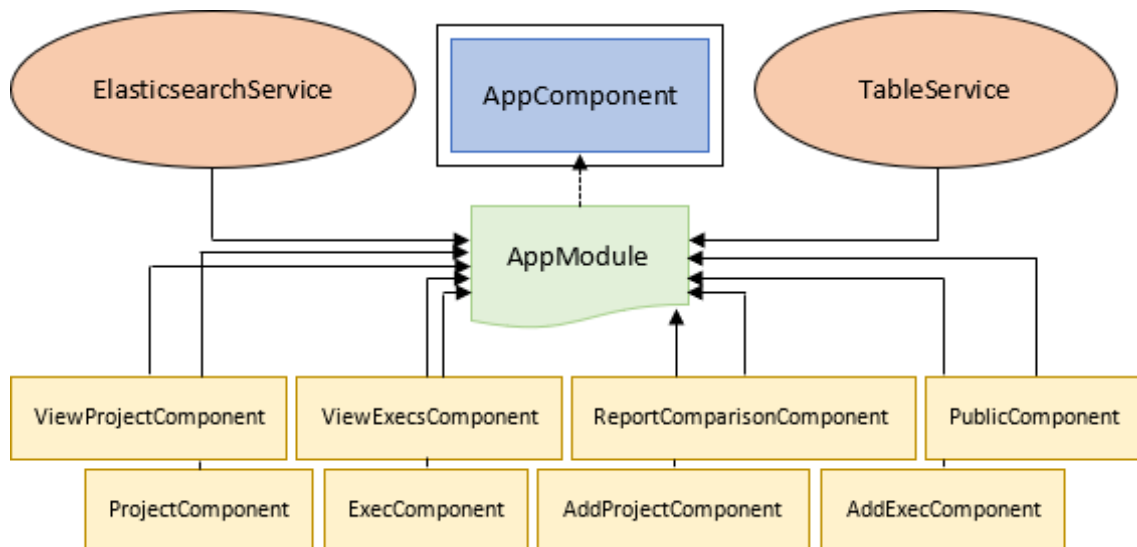


Fig. 4.8. Diagrama del front end

Además, existen dos servicios:

- ElasticsearchService. Este servicio sirve de enlace entre los componentes y la API REST y los provee de datos para mostrar información al usuario y para atender a sus peticiones. La conexión con el back end es a través de HTTP.
- TableService. Convierte la respuesta que contiene la comparación en una tabla, de tal forma que a la izquierda de la misma se visualiza el resultado para la ejecución que actúa como comparador y, a la izquierda, con la que se compara.

El front end ha sido desarrollado con TypeScript, en concreto, con el framework Angular y sus librerías de estilo asociadas: Angular Material, Teradata Covalent y NG Bootstrap, del cual se utilizó únicamente su "Popover", ya que ni Angular Material ni Teradata Covalent tenían uno en su librería.

4.3.2. Back end

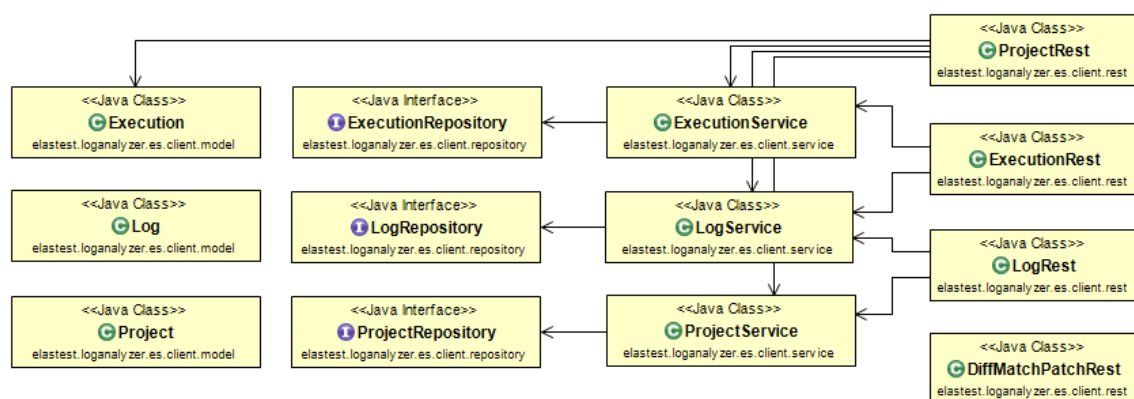


Fig. 4.9. Diagrama del back end

El back end (Fig. 4.9) está compuesto por un conjunto de controladores REST que atienden a las peticiones HTTP realizadas por el front end. Cada controlador, dispone de un repositorio y de un servicio que opera sobre ese repositorio. El repositorio, más tarde, opera sobre la base de datos de Elasticsearch para actualizar, mostrar o borrar información indexada.

El motivo de que ningún controlador acceda al repositorio directamente es para evitar accesos inseguros, buscando operar siempre con un servicio como intermediario que pueda realizar funciones previas o posteriores a la petición por parte del controlador.

4.4. Implementación

Durante el proceso de desarrollo del proyecto, muchos han sido los obstáculos que ha habido que sortear, ya sea por incompatibilidad de versiones, de productos o por la necesidad de implementar una funcionalidad que necesitaba un proceso de resolución intelectual previa a la resolución física del mismo.

4.4.1. Algoritmo de comparación

El algoritmo de comparación es el núcleo de este proyecto y, por tanto, un elemento con el que nos hemos detenido algún tiempo. El algoritmo base fue el diseñado por el desarrollador de Google, Neil Fraser⁴², quien lo bautizó bajo el nombre Diff, Match and Patch. Fue pensado para comparar un fragmento de texto plano con otro texto plano, obteniendo como salida una lista de valores, para cada caracter o cadena de caracteres, indicando eliminación (-1), adición (1) o mantenimiento (0) en la misma.

Este algoritmo [12], como salida, devolvía o una lista de diferencias, o un elemento HTML básico con las diferencias. Dado que quisimos otorgar a la aplicación un formato de salida concreto y cuidado, decidimos cambiar el método original del algoritmo que devolvía el elemento HTML por otro, que devolviese texto plano pero marcado con HTML, para, en el front end, transformarlo en una tabla de diferencias con dos columnas, una para los logs de la ejecución original (ejecución comparadora) y, la otra, para los logs de la ejecución seleccionada por el usuario (ejecución comparada). El método que obtiene una tabla de diferencias dada un elemento HTML es el siguiente:

```
1 generateTable(diff: string): any[] {  
2   this.results = [];  
3   this.comparatorClass = 'normal';  
4   this.comparedClass = 'normal';  
5   const lines = diff.split('<br>');  
6   lines.pop();  
7   let comparedLine = '', comparatorLine = '';  
8   let i = 1;
```

⁴²<https://neil.fraser.name/writing/sync/>

```

9      lines.forEach(line => {
10         line = this.closeOpenedTags(line.replace('&para;', ''));
11         line = this.openClosedTags(line);
12         comparatorLine = this.cleanBetweenTags('<ins>', '</ins>',
13             line, 0);
14         comparedLine = this.cleanBetweenTags('<del>', '</del>', line
15             , 1);
16         this.concatResults(i, comparatorLine, comparedLine);
17         i++;
18     });
19     this.solveUselessDiffs();
20     this.solveBasicTableColors();
21     return this.results;
22 }

```

El método utiliza otros métodos auxiliares de ayuda a la resolución de incongruencias o marcas abiertas.

La comparación que se realizará es semántica, diferenciando palabras en lugar de letras, para una diferenciación más precisa (si fuese una búsqueda eficiente o normal, letra a letra, el resultado reflejaría una comparación por letras). Esta comparación, para ejecuciones completas (de unas 500 líneas más o menos), se resuelve en unos 2 segundos. Para comparaciones más rápidas, se pueden modificar los parámetros de la API REST, aunque la herramienta perdería efectividad de análisis.

En cuanto a la personalización del algoritmo, puesto que no se compara texto plano sino líneas, los atributos necesarios para lograr un funcionamiento más preciso. Éstos se muestran en Tab. 4.2.

Atributo	Tipo	Función	Valor
Diff_Timeout	float	Tiempo máximo para definir una diferencia.	0.0f
Diff_EditCost	short	Coste de realizar una operación vacía (sin diferencias).	1
Match_Threshold	float	Grado de exactitud que debe darse para que dos elementos se consideren diferentes.	0.5f
Match_Distance	int	A cuánta distancia el algoritmo buscará una coincidencia. A esta distancia, el algoritmo añadirá una puntuación comprendida entre 1.0 a 0.0, siendo esta última, una coincidencia exacta.	3000
Patch_DeleteThreshold	float	Cuando se elimina un fragmento de texto (64 caracteres), cuán parecido debe de ser el contenido esperado, con un grado de exactitud determinado por Match_Threshold.	0.5f

Tab. 4.2. Atributos del algoritmo Diff, Match and Patch

Previo a la obtención de las diferencias, se mencionó en el capítulo anterior que había varios modos de comparación. Antes de enviar la petición HTTP a la API REST con las dos ejecuciones a comparar, se prepara su contenido ajustándose al modo de comparación seleccionado por el usuario. El método encargado de tal tarea es el siguiente:

```

1  private generateOutput(logs: Log[]) {
2      let result = '';
3      let comparatorDate = new Date();
4      if (logs[0] === undefined) {
5          return result;
6      }
7      for (let i = 0; i < logs.length; i++) {
8          (logs[i].timestamp.length > 2) ? (logs[i].timestamp = logs[i].
9              timestamp.substring(0, 23)) : (logs[i].timestamp = '');
10         (logs[i].thread.length > 2) ? ((logs[i].thread.indexOf('[')
11             === -1) && (logs[i].thread = '[' +
12             logs[i].thread + ']' + ')) : (logs[i].thread = '');
13         (logs[i].level.length > 2) ? (logs[i].level = logs[i].level) :
14             (logs[i].level = '');
15         (logs[i].logger.length > 2) ? (logs[i].logger = logs[i].logger
16             ) : (logs[i].logger = '');
17     }
18     if ((this.comparisonMode + '') === '2') {
19         comparatorDate = new Date(this.findValidTimestamp(logs));
20     }
21     for (let i = 0; i < logs.length; i++) {
22         ((this.comparisonMode + '') === '1') && (logs[i].timestamp = '
23             ');
24         if (((this.comparisonMode + '') === '2') && (logs[i].timestamp
25             .length > 2)) {
26             logs[i].timestamp = ((new Date(logs[i].timestamp)).valueOf()
27                 - (comparatorDate).valueOf()).toString();
28         }
29         result += (logs[i].timestamp + logs[i].thread + logs[i].level
30             + ' ' + logs[i].logger + ' ' +
31             ' ' + logs[i].message) + '\r\n';
32     }
33     return result;
34 }

```

4.4.2. Diferenciación por petición a la API REST

Uno de los aspectos más complejos consistió en dotar a la API REST de una petición que permitiese, no solo a usuarios del TestLogAnalyzer, sino a cualquiera que estuviese interesado en obtener un documento HTML con el fruto de una diferenciación, realizar una simple petición HTTP con los dos fragmentos del código a analizar y que la API devolviese, en lenguaje de marcado como HTML, el resultado de la comparación semántica.

Para la resolución de este problema, primero, creamos un nuevo RestController en el back end, al cual añadimos el método de la consulta:

```
1  @RequestMapping(value = "", method = RequestMethod.POST)
2  public ResponseEntity<String> postByDiffs(@RequestBody String body)
   throws JSONException {
3      JSONObject obj = new JSONObject(body);
4      LinkedList<Diff> d = dmp.diff_main(obj.getString("text1"),
        obj.getString("text2"));
5      dmp.diff_cleanupSemantic(d);
6      return new ResponseEntity<>(dmp.diff_prettyHtml(d),
        HttpStatus.OK);
7  }
```

En este punto, el usuario podría ser capaz de enviar una petición a la API REST con los textos a comparar en forma de objeto JSON y la API le devolvería una cadena de caracteres que se relacionaría con un elemento HTML.

4.4.3. Tratamiento de múltiples ficheros (.txt y .xml)

La funcionalidad que TestLogAnalyzer ofrece apoya la subida y tratamiento de varios ficheros .txt y .xml a la vez, de una forma asombrosamente rápida. El método encargado de ello se encuentra en la API REST, dentro del controlador ProjectRest. Este método, primero, recorre los documentos de texto, ingstando los logs utilizando expresiones regulares a la base de datos. Después, se parsean los documentos del Surefire Report Plugin del siguiente modo:

```
1  TestSuiteXmlParser parser = new TestSuiteXmlParser(consoleLogger);
2      InputStream inputStream = file.getInputStream();
3      InputStreamReader inputStreamReader = new InputStreamReader(
        inputStream, "UTF-8");
4      List<ReportTestSuite> tests = parser.parse(inputStreamReader);
5      for (int j = 0; j < tests.size(); j++) {
6          this.execution.setErrors(this.execution.getErrors() + tests.
            get(j).getNumberOfErrors());
7          this.execution.setFailures(this.execution.getFailures() +
            tests.get(j).getNumberOfFailures());
8          this.execution.setFlakes(this.execution.getFlakes() + tests.
            get(j).getNumberOfFlakes());
9          this.execution.setSkipped(this.execution.getSkipped() +
            tests.get(j).getNumberOfSkipped());
10         this.execution.setTests(this.execution.getTests() + tests.
            get(j).getNumberOfTests());
11         List<ReportTestCase> testcases = this.execution.getTestcases
            ();
12         testcases.addAll(tests.get(j).getTestCases());
13         this.execution.setTestcases(testcases);
```

```
14         this.execution.setTime_elapsed(this.execution.  
15         getTime_elapsed() + tests.get(j).getTimeElapsed());  
    }
```

4.4.4. Integración continua sobre la aplicación

El segundo detalle más complejo fue al final del desarrollo del proyecto, cuando tanto el front como el back end compartían repositorio. Debido a que estábamos utilizando un desarrollo iterativo e incremental, era fundamental para nosotros mantener la calidad de los tests y un entorno que los ejecutase con cada commit. Las dos herramientas que utilizábamos que ofrecían esta posibilidad eran Travis CI y CircleCI.

CircleCI fue descartado, ya que esta herramienta no permitía dos entornos de ejecución en un mismo repositorio. En su lugar, utilizamos TravisCI e integramos las tareas ejecutadas por CircleCI (Travis ejecutaba los tests y Circle analizaba el código con yarn). Tras varios intentos, el script fue el siguiente:

```
1  sudo: required  
2  matrix:  
3    include:  
4      - language: java  
5        jdk: openjdk8  
6        services:  
7          - elasticsearch  
8        before_install:  
9          - curl -O https://download.elastic.co/elasticsearch/  
            release/org/elasticsearch/distribution/deb/  
            elasticsearch/2.4.6/elasticsearch-2.4.6.deb \&\&  
            sudo dpkg -i --force-confnew elasticsearch  
            -2.4.6.deb \&\& sudo service elasticsearch restart  
10       before_script:  
11         - cd ./testloganalyzer  
12         - sleep 100  
13       script:  
14         - mvn clean install -Dspring.profiles.active=dev  
15       after_script:  
16         - cd ..  
17  
18     - language: node_js  
19       node_js: 8  
20       addons:  
21         apt:  
22           sources:  
23             - google-chrome  
24           packages:  
25             - google-chrome-stable  
26             - google-chrome-beta  
27       before_install:
```

```
28     - export CHROME_BIN=chromium-browser
29     - export DISPLAY=:99.0
30     - sh -e /etc/init.d/xvfb start
31     before_script:
32     - cd ./testlogalyzer-gui
33     - npm install -g @angular/cli
34     - npm install -g yarn
35     - npm install
36     script:
37     - yarn lint
38     - ng test
39     - ng build
40     after_script:
41     - cd ..
```

En este script, Travis CI crea dos entornos de pruebas, uno para el front end y otro para el back end. Para cada uno de ellos, ejecuta los tests y, para el caso del front end, ejecuta el análisis de código de Yarn.

4.5. Pruebas

Puesto que este proyecto tiene como objetivo analizar los resultados obtenidos en la ejecución de tests de un proyecto y realizar comparaciones, no podían faltar las pruebas sobre este producto.

4.5.1. Implementación

Para la implementación de pruebas del front end y del back end no se han utilizado clientes especiales para ello, sino que se ha recurrido al empleo de las propias herramientas que ofrecen los entornos de desarrollo de cada una de las tecnologías utilizadas. El front end contiene un total de 12 tests y el back end, un total de 43 tests.

Front end

Cada uno de los elementos del front end, sin contar los modelos, están dotados de un archivo de prueba, que, en lugar de tener una extensión .ts, tienen una extensión .spec.ts. Para todos los componentes, estos tests prueban funciones críticas de ellos (por ejemplo, si el componente tiene un botón que realiza una función como eliminar un elemento o acceder a él, se prueba).

Además, todos los componentes tienen una función común que determina si han sido creados de forma satisfactoria:

```
1 it('Should create the component', () => {
```



```
2     const app = fixture.debugElement.componentInstance;
3     expect(app).toBeTruthy();
4   });
```

Para todas estas pruebas, se utiliza la clase TestBed que permite configurar e inicializar el entorno para realizar tests unitarios y provee de métodos para crear componentes y servicios durante la ejecución de esos tests.

Estos tests han sido ejecutados por TravisCI de forma automática en la publicación de cada commit. Ahondaremos en este detalle en la próxima sección.

Back end

Hemos tenido especial cuidado a la hora de probar el back end, ya que es el elemento más importante de TestLogAnalyzer, puesto que contiene toda la lógica computacional del proyecto. Menos los modelos y los servicios (ya que, si probamos los repositorios, probar los servicios sería redundante), el resto de elementos han sido probados utilizando tests unitarios.

Para probar los repositorios, hemos realizado, para cada modelo, peticiones siguiendo el patrón GWT:

```
1  @Test
2  public void shouldReturnAddedId() {
3      // Given
4      Execution e1 = new Execution(99999998, 201, 23, 2, 34, "
          JUnit4ClassTestingTLA", 0, new Date().toString(), "BUILD
          SUCCESS", 3, new ArrayList<ReportTestCase>(), (float) 20.3);
5      Execution e2 = new Execution(99999999, 43, 3, 4, 0, "
          JUnit4ClassTestingTLA", 3, new Date().toString(), "BUILD
          FAILURE", 2, new ArrayList<ReportTestCase>(), (float) 603.2);
6      // When
7      Execution returnedValue1 = repository.save(e1);
8      Execution returnedValue2 = repository.save(e2);
9      // Then
10     assertEquals(returnedValue1, e1);
11     assertEquals(returnedValue2, e2);
12     repository.delete(e1);
13     repository.delete(e2);
14 }
```

Nótese que, para evitar añadir basura a la base de datos, antes de probar una funcionalidad añadimos unos elementos de prueba, y, tras probarla, los eliminamos.

Para la creación de tests para los controladores de la API REST, hemos utilizado un mocks que nos permitan lanzar peticiones HTTP:

```

1  @Test
2  public void getById() throws Exception {
3      int id = 0;
4      mockMvc.perform(get("/api/executions").param("id", String.valueOf
        (id)));
5  }

```

4.5.2. Automatización

Los tests se ejecutan automáticamente tras cada commit (Fig. 4.10), sea cual sea el branch y sea cual sea el contenido de ese commit. Partiendo de la premisa de que estamos desarrollando un producto en un entorno iterativo, entonces cada tarea se creará en una rama apartada de la rama principal. Cuando esta tarea haya sido desarrollada con éxito y los entornos de integración continua lo consideren (todos los tests hayan pasado), se fusionará con la principal, enriqueciendo la funcionalidad y sin deteriorar la anterior ya existente.

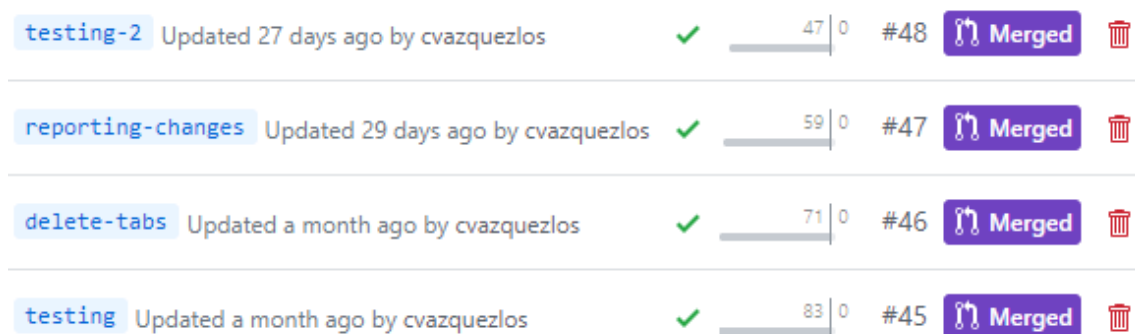


Fig. 4.10. Administración de las ramas del repositorio

5. CONCLUSIONES

5.1. Objetivos cumplidos

No cabe duda que, con TestLogAnalyzer, se ha creado una herramienta que permite gestionar y analizar ejecuciones y proyectos. Además, como módulo del proyecto ElasTest, se ha enriquecido la funcionalidad total que esta herramienta proporciona a sus usuarios, además de facilitar una conexión indirecta entre los entornos de integración continua conocidos y TestLogAnalyzer, gracias a la creación de una API REST que admite peticiones cURL.

En adición a lo anterior, se han aplicado los conocimientos obtenidos en las asignaturas que trataban sobre el desarrollo ágil e iterativo: Procesos del Software y Calidad del Software. También, se ha desarrollado un producto autónomo y que responde correctamente a los cambios aplicando las nociones de Evolución y Adaptación del Software, así como desarrollarlo con una interfaz gráfica accesible e intuitiva, siguiendo los criterios de Interacción Persona-Ordenador.

Las reuniones periódicas (una cada 2 ó 3 semanas), el versionado, la utilización de herramientas de integración continua y de control de versiones y una comunicación fluida, hicieron que el desarrollo de este proyecto haya sido un proyecto extremadamente cercano al ámbito profesional, con la excepción de que el equipo de desarrollo estaba conformado por un individuo.

5.2. Líneas futuras de trabajo

Queda pendiente, conforme avance la tecnología, actualizar el proyecto a las versiones más punteras. A pesar de que, a fecha de hoy, el producto ha sido desarrollado con las últimas versiones para evitar ineficiencias o fallos que corrigen los versionados, es por todos conocido que la tecnología avanza con extrema rapidez, y, pronto, el producto se encontrará en un estado de desactualización. Para impedir esta situación, el equipo de ElasTest será el encargado de su mantenimiento, aunque la forma de desarrollo y los patrones utilizados facilitan esta labor.

Concretando ligeramente, un aspecto que se quedó sin implementar y que el tutor de este TFG tuvo en mente durante todo el desarrollo, pero que, por cuestiones de tiempo, no pudo ser posible, fue la capacidad de que el algoritmo de diferenciación interprete ciertos patrones en los logs de las ejecuciones y los diera como buenos, aunque su contenido fuese distinto. Esto permitiría a la herramienta eliminar ruido y no detectar diferencias para elementos que no brindan ningún tipo de información (por ejemplo, aunque el timestamp de cada log es distinto, el algoritmo debería ignorar ese elemento).

BIBLIOGRAFÍA

- [1] C. Jones, *Applied Software Measurement*, ser. Software Productivity Research. McGraw-Hill, 1991.
- [2] D. Pogue, “5 Most Embarrassing Software Bugs in History,” *Scientific American*, 2014.
- [3] L. TypeScript. TypeScript - JavaScript that scales. [Online]. Available: <https://www.typescriptlang.org/>
- [4] AlexSoft. The Good and the Bad of Angular Development. [Online]. Available: <https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-angular-development/>
- [5] Wikipedia. Node.js. [Online]. Available: <https://es.wikipedia.org/wiki/Node.js>
- [6] ——. Objeto (programación). [Online]. Available: <https://es.wikipedia.org/wiki/Objeto>
- [7] ——. Dependencias de software. [Online]. Available: https://es.wikipedia.org/wiki/Dependencias_de_software
- [8] ——. Maven. [Online]. Available: <https://es.wikipedia.org/wiki/Maven>
- [9] ——. Contenedores de software. [Online]. Available: https://es.wikipedia.org/wiki/Contenedores_de_software
- [10] P. Ágiles. Desarrollo iterativo e incremental. [Online]. Available: <https://proyectosagiles.org/desarrollo-iterativo-incremental>
- [11] J. Garzás. El ciclo de vida iterativo e incremental y el riesgo de olvidarse del iterativo y quedarse solo con el incremental. [Online]. Available: <http://www.javiergarzas.com/2012/10/iterativo-e-incremental.html>
- [12] Google. Diff Match Patch. [Online]. Available: <https://github.com/google/diff-match-patch>

APÉNDICE A. GLOSARIO

API	Application Programming Interface
CI	Continuous Integration
CLI	Command-Line Interface
CLM	Cloud Log Miner
E2E	End to End
ELK	Elasticsearch, Logstash & Kibana
GWT	Given, When, Then
GUI	Graphic User Interface
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
I/O	Input/Output
IDE	Integrated Development Environment
JS	JavaScript
JSON	JavaScript Object Notation
MVC	Model-View-Controller
NG	Next Generation
OS	Operating System
POM	Project Object Model
REST	Representational State Transfer
SPA	Single Page Application
TDD	Test-Driven Development
TFG	Trabajo de Fin de Grado
TXT	Plain TeXT
XML	EXtensive Markup Language

APÉNDICE B. EJEMPLO DE DOCUMENTO DE EJECUCIÓN DE TESTS (.TXT)

```
1 [INFO]
2 [INFO] --- maven-resources-plugin:2.6:testResources (default-testResources) @ full-teaching ---
3 [INFO] Using 'UTF-8' encoding to copy filtered resources.
4 [INFO] Copying 2 resources
5 [INFO]
6 [INFO] --- maven-compiler-plugin:3.1:testCompile (default-testCompile) @ full-teaching ---
7 [INFO] Nothing to compile - all classes are up to date
8 [INFO]
9 [INFO] --- maven-surefire-plugin:2.18.1:test (default-test) @ full-teaching ---
10 [INFO] Surefire report directory: /home/pablo/Desktop/full-teaching-experiment/target/surefire-reports
11
12 -----
13 T E S T S
14 -----
15 Running com.fullteaching.backend.e2e.FullTeachingTestE2EChat
16 2018-04-09 14:25:06.823 INFO --- [ main] io.github.bonigarcia.wdm.BrowserManager : Reading https://
   chromedriver.storage.googleapis.com/ to seek [chromedriver]
17 2018-04-09 14:25:07.389 INFO --- [ main] io.github.bonigarcia.wdm.BrowserManager : Latest version of [
   chromedriver] is 2.37
18 2018-04-09 14:25:07.397 INFO --- [ main] io.github.bonigarcia.wdm.BrowserManager : Exporting webdriver.
   chrome.driver as /home/pablo/.m2/repository/webdriver/chromedriver/linux64/2.37/chromedriver
19 2018-04-09 14:25:11.991 INFO --- [ main] c.f.backend.e2e.FullTeachingTestE2E : ##### Start test:
   oneToOneChatInSessionChrome
20 2018-04-09 14:25:11.991 INFO --- [ main] c.f.backend.e2e.FullTeachingTestE2E : Starting browser (chrome)
21 2018-04-09 14:25:12.670 INFO --- [ main] c.f.backend.e2e.FullTeachingTestE2E : Navigating to https://
   localhost:5000/
22 2018-04-09 14:25:14.971 INFO --- [ main] c.f.backend.e2e.FullTeachingTestE2E : Logging in user Teacher
   with mail 'teacher@gmail.com'
23 2018-04-09 14:25:19.298 INFO --- [ main] c.f.backend.e2e.FullTeachingTestE2E : Logging in successful for
   user Teacher
24 2018-04-09 14:25:20.298 INFO --- [ main] c.f.backend.e2e.FullTeachingTestE2E : Teacher entering first
   course
25 2018-04-09 14:25:21.472 INFO --- [ main] c.f.backend.e2e.FullTeachingTestE2E : Teacher navigating to '
   Sessions' tab
26 2018-04-09 14:25:22.634 INFO --- [ main] c.f.backend.e2e.FullTeachingTestE2E : Teacher getting into
   first session
27 2018-04-09 14:25:23.870 INFO --- [ main] c.f.backend.e2e.FullTeachingTestE2E : Checking system message (
   "Connected") for Teacher
28 2018-04-09 14:25:24.452 INFO --- [ main] c.f.backend.e2e.FullTeachingTestE2E : Starting browser (chrome)
29 2018-04-09 14:25:24.819 INFO --- [ main] c.f.backend.e2e.FullTeachingTestE2E : Navigating to https://
   localhost:5000/
30 2018-04-09 14:25:27.552 INFO --- [ main] c.f.backend.e2e.FullTeachingTestE2E : Logging in user Student
   with mail 'student1@gmail.com'
31 2018-04-09 14:25:31.519 INFO --- [ main] c.f.backend.e2e.FullTeachingTestE2E : Logging in successful
32 2018-04-09 14:25:36.058 INFO --- [ main] c.f.backend.e2e.FullTeachingTestE2E : Checking system message (
   "Student Imprudent has connected") for Teacher
33 2018-04-09 14:25:36.139 INFO --- [ main] c.f.backend.e2e.FullTeachingTestE2E : Checking system message (
   "Teacher Cheater has connected") for Student
34 2018-04-09 14:25:38.776 INFO --- [ main] c.f.backend.e2e.FullTeachingTestE2E : Checking own message ("
   TEACHER CHAT MESSAGE") for Teacher
35 2018-04-09 14:25:40.809 INFO --- [ main] c.f.backend.e2e.FullTeachingTestE2E : Checking own message ("
   STUDENT CHAT MESSAGE") for Student
36 2018-04-09 14:25:42.957 INFO --- [ main] c.f.backend.e2e.FullTeachingTestE2E : Logging out Student
37 2018-04-09 14:25:44.760 INFO --- [ main] c.f.backend.e2e.FullTeachingTestE2E : Logging out successful
   for Student
38 2018-04-09 14:25:45.934 INFO --- [ main] c.f.backend.e2e.FullTeachingTestE2E : Logging out Teacher
39 2018-04-09 14:25:47.419 INFO --- [ main] c.f.backend.e2e.FullTeachingTestE2E : Logging out successful
   for Teacher
40 2018-04-09 14:25:48.552 INFO --- [ main] c.f.backend.e2e.FullTeachingTestE2E : ##### Finish test:
   oneToOneChatInSessionChrome
41 Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 42.52 sec - in com.fullteaching.backend.e2e.
   FullTeachingTestE2EChat
42
43 Results :
44
45 Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
46
47 [INFO] -----
48 [INFO] BUILD SUCCESS
49 [INFO] -----
50 [INFO] Total time: 44.704 s
51 [INFO] Finished at: 2018-04-09T14:25:48+02:00
52 [INFO] Final Memory: 27M/482M
53 [INFO] -----
```

APÉNDICE C. EJEMPLO DE DOCUMENTO DE SUREFIRE REPORT PLUGIN (.XML)

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <testsuite name="com.fullteaching.backend.e2e.FullTeachingTestE2EChat" time="36.583" tests="1" errors="0" skipped
  ="0" failures="0">
3   <properties>
4     <property name="java.runtime.name" value="OpenJDK Runtime Environment"/>
5     <property name="sun.boot.library.path" value="/usr/lib/jvm/java-8-openjdk-amd64/jre/lib/amd64"/>
6     <property name="java.vm.version" value="25.162-b12"/>
7     <property name="java.vm.vendor" value="Oracle Corporation"/>
8     <property name="maven.multiModuleProjectDirectory" value="/home/pablo/Desktop"/>
9     <property name="java.vendor.url" value="http://java.oracle.com"/>
10    <property name="path.separator" value=":"/>
11    <property name="guice.disable.misplaced.annotation.check" value="true"/>
12    <property name="java.vm.name" value="OpenJDK 64-Bit Server VM"/>
13    <property name="file.encoding.pkg" value="sun.io"/>
14    <property name="user.country" value="US"/>
15    <property name="sun.java.launcher" value="SUN_STANDARD"/>
16    <property name="sun.os.patch.level" value="unknown"/>
17    <property name="test" value="FullTeachingTestE2EChat"/>
18    <property name="java.vm.specification.name" value="Java Virtual Machine Specification"/>
19    <property name="user.dir" value="/home/pablo/Desktop/full-teaching-experiment"/>
20    <property name="java.runtime.version" value="1.8.0_162-8u162-b12-0ubuntu0.16.04.2-b12"/>
21    <property name="java.awt.graphicsenv" value="sun.awt.X11GraphicsEnvironment"/>
22    <property name="java.endorsed.dirs" value="/usr/lib/jvm/java-8-openjdk-amd64/jre/lib/endorsed"/>
23    <property name="os.arch" value="amd64"/>
24    <property name="java.io.tmpdir" value="/tmp"/>
25    <property name="line.separator" value="&#10;"/>
26    <property name="java.vm.specification.vendor" value="Oracle Corporation"/>
27    <property name="os.name" value="Linux"/>
28    <property name="classworlds.conf" value="/usr/share/maven/bin/m2.conf"/>
29    <property name="sun.jnu.encoding" value="UTF-8"/>
30    <property name="java.library.path" value="/usr/java/packages/lib/amd64:/usr/lib/x86_64-linux-gnu:/usr/lib/jni:/lib:/usr/lib/
      x86_64-linux-gnu:/usr/lib/x86_64-linux-gnu:/usr/lib/jni:/lib:/usr/lib"/>
31    <property name="java.specification.name" value="Java Platform API Specification"/>
32    <property name="java.class.version" value="52.0"/>
33    <property name="sun.management.compiler" value="HotSpot 64-Bit Tiered Compilers"/>
34    <property name="os.version" value="4.13.0-38-generic"/>
35    <property name="user.home" value="/home/pablo"/>
36    <property name="user.timezone" value="Europe/Madrid"/>
37    <property name="java.awt.printerjob" value="sun.print.PSPrinterJob"/>
38    <property name="java.specification.version" value="1.8"/>
39    <property name="file.encoding" value="UTF-8"/>
40    <property name="user.name" value="pablo"/>
41    <property name="java.class.path" value="/usr/share/maven/boot/plexus-classworlds-2.x.jar"/>
42    <property name="java.vm.specification.version" value="1.8"/>
43    <property name="sun.arch.data.model" value="64"/>
44    <property name="java.home" value="/usr/lib/jvm/java-8-openjdk-amd64/jre"/>
45    <property name="sun.java.command" value="org.codehaus.plexus.classworlds.launcher.Launcher -Dtest=
      FullTeachingTestE2EChat -B test"/>
46    <property name="java.specification.vendor" value="Oracle Corporation"/>
47    <property name="user.language" value="en"/>
48    <property name="awt.toolkit" value="sun.awt.X11.XToolkit"/>
49    <property name="java.vm.info" value="mixed mode"/>
50    <property name="java.version" value="1.8.0_162"/>
51    <property name="java.ext.dirs" value="/usr/lib/jvm/java-8-openjdk-amd64/jre/lib/ext:/usr/java/packages/lib/
      ext"/>
52    <property name="securerandom.source" value="file:/dev/./urandom"/>
53    <property name="sun.boot.class.path" value="/usr/lib/jvm/java-8-openjdk-amd64/jre/lib/resources.jar:/usr/lib/
      jvm/java-8-openjdk-amd64/jre/lib/rt.jar:/usr/lib/jvm/java-8-openjdk-amd64/jre/lib/sunrsasn.jar:/usr/
      lib/jvm/java-8-openjdk-amd64/jre/lib/jsse.jar:/usr/lib/jvm/java-8-openjdk-amd64/jre/lib/jce.jar:/usr/lib
      /jvm/java-8-openjdk-amd64/jre/lib/charsets.jar:/usr/lib/jvm/java-8-openjdk-amd64/jre/lib/jfr.jar:/usr/
      lib/jvm/java-8-openjdk-amd64/jre/classes"/>
54    <property name="java.vendor" value="Oracle Corporation"/>
55    <property name="maven.home" value="/usr/share/maven"/>
56    <property name="file.separator" value=""/>
57    <property name="java.vendor.url.bug" value="http://bugreport.sun.com/bugreport"/>
58    <property name="sun.cpu.endian" value="little"/>
59    <property name="sun.io.unicode.encoding" value="UnicodeLittle"/>
60    <property name="sun.desktop" value="gnome"/>
61    <property name="sun.cpu.isalist" value=""/>
62  </properties>
63  <testcase name="oneToOneChatInSessionChrome(TestInfo)" classname="E2E tests for FullTeaching chat" time="36.583
    "/>
64 </testsuite>
```