ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

GRADO EN INGENIERÍA DEL SOFTWARE

Curso académico 2017/2018

Trabajo de Fin de Grado

TESTLOGANALYZER: Análisis de logs

Autor: Carlos Vázquez Losada

Tutor: Micael Gallego Carrillo

Contenido

1. Introducción y Motivación

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 émulas 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.

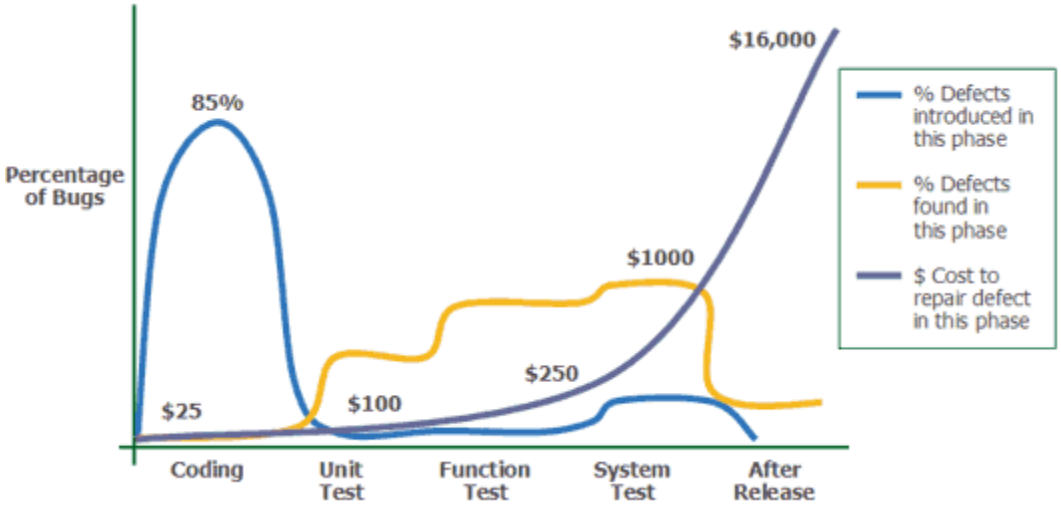


Fig. 1. Evolución económica de la no resolución de cierta cantidad de bugs.

En 1996, Capers Jones [1] hacía explícito, tal y como se muestra en Fig. 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 robóta 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.

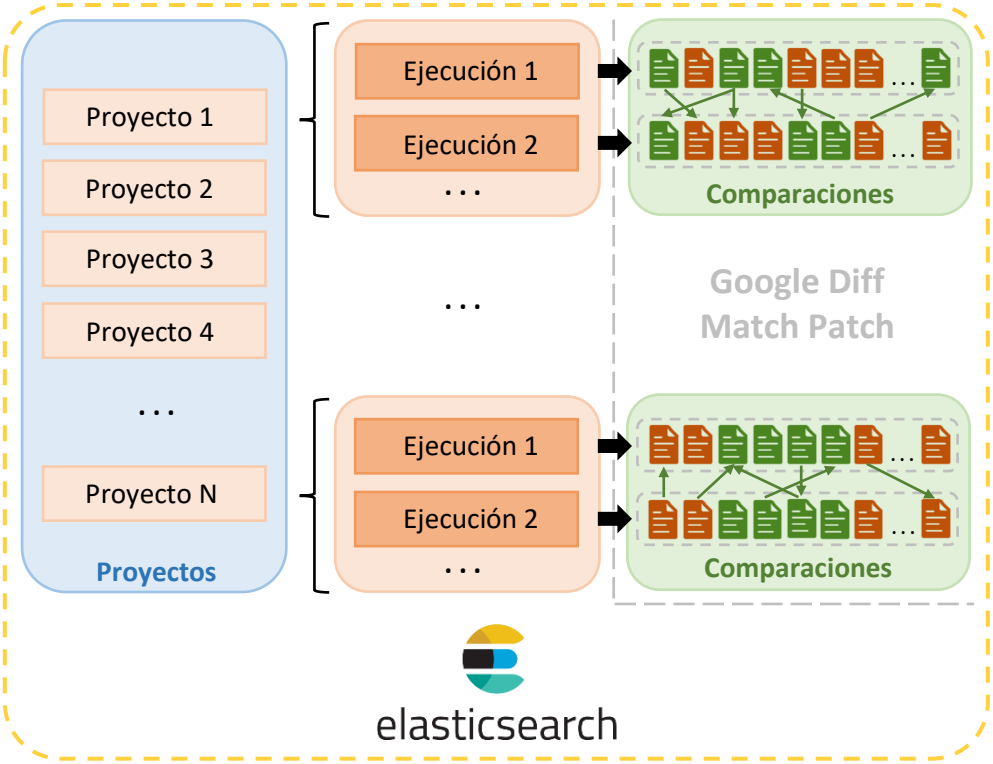
TestLogAnalyzer permite almacenar, consultar y comparar ejecuciones de tests finalizados, ya hayan concluido como erróneos o exitosos, para buscar cambios entre ellos. Lo anterior unido al criterio del usuario, es la herramienta perfecta para encontrar diferencias que haya llevado a una ejecución a fallar con respecto a anteriores que finalizaron correctamente y viceversa.

2. Objetivos

El objetivo principal de este proyecto será realizar un nuevo componente para ElasTest, plataforma para labores de test en complejos sistemas de software distribuidos y de gran tamaño.

TestLogAnalyzer cubre la necesidad de analizar los logs provenientes de los distintos subsistemas de una aplicación. Así, ElasTest permitirá, una vez este proyecto haya sido integrado, crear proyectos, compuestos por un número concreto de subsistemas, y, a cada subsistema, se le podrá agregar sus correspondientes logs. Después, estos logs pueden ser visualizados y comparados entre sí, apoyándose en un algoritmo de diferenciación.

Este algoritmo es el núcleo del proyecto y parte del creado por Neil Fraser, Google Diff Match Patch, con algunas modificaciones, para realizar esta comparativa semántica:



Como se puede apreciar, las comparaciones entre logs se podrán realizar en el ámbito del proyecto al que pertenecen, pudiéndose comparar logs de distintos componentes. Esto ha sido diseñado así debido a que la comparación entre proyectos no tiene sentido alguno, ya que se presupone que el usuario debe organizar su proyecto tal y como se indica, integrando los subsistemas de un sistema dentro de un proyecto.

TestLogAnalyzer tiene, por tanto, una estructura lógica, acorde con ElasTest, y que permite que el usuario añada sus propios proyectos y cree sus propios subsistemas a través de un sistema de pestañas muy intuitivo, facilitando la dificultosa labor del mundo del testing.

3. Tecnologías, Herramientas y Metodologías

3.1. Tecnologías

Debido a que partíamos de un proyecto ya existente, tal y como se ha mencionado con anterioridad, la tarea de seleccionar el lenguaje de programación para llevar a cabo el proyecto no ha sido muy compleja, ya que el proyecto obsoleto estaba realizado con la antiquísima versión Angular 2 y no contaba con un cliente back end.

3.1.1. TypeScript

Versión 2.5.3

Ilustración 1. Logo TypeScript

TypeScript se define a sí mismo como [3] "*JavaScript that scales. TypeScript is a typed superset of JavaScript that compiles to plain JavaScript in any browser, any host and any OS*".

TypeScript parte de *JavaScript*, 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.

En **TestLogAnalyzer**, elegimos *TypeScript* porque:

- **Parte de *JavaScript***. *JavaScript* es un lenguaje ampliamente conocido por todo el mundo y al que los desarrolladores recurren si tienen la oportunidad. Es de rápido aprendizaje, tiene una grandísima cantidad de librerías y está en constante evolución.
- **Permite desarrollar grandes aplicaciones**. Los tipos permiten la utilización de herramientas y metodologías altamente productivas, como la comprobación la comprobación estática y la refactorización del código cuando se desarrollan aplicaciones en *JavaScript*.

En definitiva; *TypeScript* es un lenguaje rápido, tipado y pensado para aplicaciones escalables, ámbito completamente aprovechado por **TestLogAnalyzer** y por las herramientas implementadas con este lenguaje.

3.1.1.1. Angular

Ya que este proyecto comenzó su desarrollo en agosto de 2017, momento en el que la actual versión de Angular era la 4, hasta el mes de diciembre de este mismo año la seguimos utilizando, hasta que, en diciembre, salió a la luz la versión estable de Angular 5.

Versión 5.1.2

Ilustración 2. Logo Angular

Angular 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* (*Single Page Application*) 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.

Puesto que está basado en *TypeScript*, está dotado de una gran cantidad de herramientas y permite una elaboración de un código limpio y de amplia escalabilidad.

Sin embargo, [4] cabe mencionar que, para programadores poco experimentados, Angular puede ser un mal punto de partida, dado que es engorroso y complejo. Por ejemplo, son necesarios más de cinco archivos para poder tener un componente operativo. En consecuencia, la mayor parte del desarrollo de un proyecto Angular consiste en realizar tareas repetitivas y la curva de aprendizaje es complicada incluso para experimentados en *JS*.

3.1.1.2. Angular CLI

Versión 1.6.3

Ilustración 3. Logo Angular CLI

Angular CLI 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 *Karma*, *Jasmine* o *e2e*. Las dependencias de un proyecto Angular también serán administradas por esta interfaz, que provee de comandos para ejecutar la aplicación, crear nuevos componentes, actualizar las dependencias…

La sencillez con la que fue desarrollada esta herramienta junto con su fácil uso, hace que programadores más experimentados no puedan acceder a una configuración con aspectos más avanzados.

En definitiva; *Angular CLI* hace que el proyecto sea más fácil de administrar, pero lo hace menos flexible, orientado a principiantes y proyectos de baja envergadura, haciéndose inevitable la participación manual para proyectos que requieran una configuración más avanzada.

Esta herramienta nos fue altamente útil cuando intentamos lanzar el CLM (*Cloud Log Miner*), ya que estaba desactualizado y necesitábamos una versión antigua de Angular. Con el CLI, pudimos crear el proyecto y ejecutarlo para comprobar la funcionalidad de nuestra compañera Silvia, que realizó el *Cloud Log Miner* del que parte este proyecto.

3.1.1.3. Angular Material

Fue el 25 de junio de 2014 cuando Google lanzó una directiva de diseño enfocado en la visualización de Android, además de en la web y en cualquier plataforma de las aplicaciones que corren sobre ellas. Esta directiva recibe el nombre de *Material Design*.

Versión 5.0.2

Ilustración 4. Logo Angular Material

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

3.1.1.4. Teradata Covalent

Versión 1.0.0

Ilustración 5. Logo Teradata Covalent

Teradata Covalent 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.

3.1.1.5. NG Bootstrap

Versión 1.1.2

Ilustración 6. Logo NG Bootstrap

NG Bootstrap 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. Además, es de código abierto y hay una grandísima comunidad detrás colaborando, además de un equipo íntegro y único centrado en este proyecto.

En concreto, hemos utilizado *NG Bootstrap* para la creación de *popups* accesibles y bien diseñados, ya que las librerías independientes que ofrecía *npm* no eran del todo funcionales y, por tanto, perdían utilidad.

3.1.2. Node.js

Versión 8.9.4

Ilustración 7. Logo Node.js

Node.js es un entorno de ejecución para *JavaScript* construido con el motor de *JavaScript V8* de Chrome. Utiliza un modelo de operaciones E/S 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**. Entorno de ejecución para *JavaScript* de Google Chrome. Escrito en *C++*, compila el código *JavaScript* en máquina en lugar de interpretarlo en tiempo real.

En definitiva: *Node.js* nos permite realizar aplicaciones que nunca se quedarán dormidas ni paradas, ya que se basa en el concepto de callback, y siempre estará a la espera de nuevas peticiones a las que atender. Además, su procesamiento es increíblemente rápido.

3.1.3. 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*.

Versión 8

Ilustración 8. Logo Java

Java 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.

3.1.3.1. Spring Boot

Versión 1.5.10

Ilustración 9. Logo Spring Boot

Spring Boot 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 *Tomcat*, *Jetty* o *Undertow* 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).

Además, *Spring* cuenta con una gran cantidad de paquetes que facilitan la vida al desarrollador:

- **Spring Data**. Simplifica el acceso y la adición de datos en bases relacionales y no relacionales.
- **Spring Security**. Plataforma de seguridad robusta.

Spring Boot facilita enormemente la utilización y adición de estos paquetes a los proyectos de *Spring*, lo que lo convierte en una gran elección.

3.2. 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.

3.2.1. 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.

3.2.1.1. Git

Versión 2.17.0

Ilustración 10. Logo de Git

Git 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…

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 Desktop.

Versión 1.1.1

Ilustración 11. Logo de GitHub (izquierda) y GitHub Desktop (derecha)

3.2.2. 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.

3.2.2.1. Tidelift

Ilustración 12. Logo Tidelift

Tidelift, 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 subscripción para empresas.

Esta herramienta examinaba las dependencias de todo el proyecto, reconociendo automáticamente entornos de front 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.

Tuvimos suerte, pues pudimos utilizarla en los primeros meses de desarrollo de este proyecto, que fue el momento más oportuno, ya que en estos primeros meses son en los que más dependencias se añaden y más control hay que tener sobre las mismas. Tras su posterior descarte, la alternativa que tuvimos fue que, al instalar una nueva dependencia en el front o en el back end, debíamos prestar especial anterior a cualquier *warning* o *error*, para impedir que el correcto funcionamiento del proyecto se viese comprometido.

Versión 6.0.0

Ilustración 13. Logo npm

npm es el administrador de paquetes para *JavaScript*. Podríamos considerar a *npm* como una herramienta que permite la colaboración entre los desarrolladores:

- 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.

Tiene una versión gratuita que, para el ámbito de este proyecto, es más que suficiente. Sin embargo, las opciones de pago escalan hasta los 16 USD, mientras que la herramienta

anterior tenía planes desde los 880 USD mensuales, lo cual lo convierte en una herramienta a utilizar sin ninguna duda.

Todos los paquetes del proyecto de *Angular* han sido instalados por *npm* y, en adición, *npm* es utilizado en aplicaciones *Node.js*.

3.2.2.3. Maven

Versión 3.5.3

Ilustración 14. Logo Maven

Maven es el gestor de dependencias para proyectos *Java*. Además, actúa como constructor de proyectos generando un archivo de configuración denominado *POM*.

El *POM* [8] (*Project Object Model*) 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).

3.2.3. 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.

3.2.3.1. WebStorm

Versión 2017.3.4

Ilustración 15. Logo WebStorm

WebStorm 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*.

Además, puesto que lo que buscamos es un desarrollo robusto y con una cobertura de *tests* amplia, *WebStorm* se convierte en una buena opción, ya que permite hacer *debugs* y ejecutar tests con *Karma*, *Mocha* y otros tipos directamente en el IDE. También, integra gestores de control de versiones como *GitHub* o *Mercurial*, entre otros.

3.2.3.2. Spring Tool Suite

Versión 3.9.2

Ilustración 16. Logo Spring Tool Suite

Spring Tool Suite 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* sin ningún inconveniente. Soporta aplicaciones de ámbito local, virtual y basadas en la nube.

Como *Eclipse*, tiene un asistente de código que facilita la utilización de los paquetes de *Spring* instalados y utilizados y un editor gráfico.

3.2.4. Entornos de integración continua

Debido a la metodología seguida, antes de implementar cualquier funcionalidad, ésta se traducía a tests, conformando un TDD. Dado este proceso, resulta indispensable la utilización de herramientas de integración contina y de análisis de código.

3.2.4.1. Travis CI

Versión 2.2

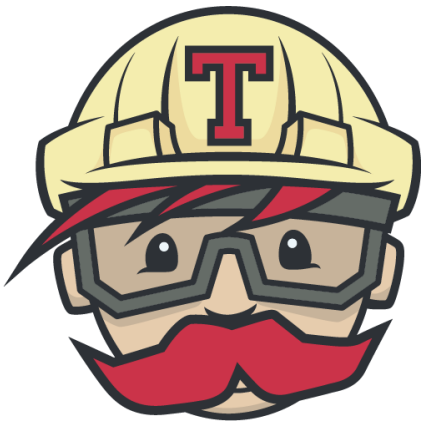


Ilustración 17. Logo Travis CI

Travis CI es un entorno de integración continua que se sincroniza con los repositorios de *GitHub* y prueba el código en cuestión de minutos. Contiene una grandísima cantidad de plataformas de desarrollo y de bases de datos que permite construir un archivo de configuración ajustado a las necesidades de los desarrolladores.

En nuestro caso particular, al alojar tanto el front como el back end en un mismo repositorio, tuvimos que realizar un amplio archivo de configuración basándonos en el concepto de *matrix* de *Travis*.

3.2.4.2. CircleCI

Versión 2.0

Ilustración 18. Logo CircleCI

CircleCI 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.

Hasta que decidimos dejar de utilizar esta herramienta, el desarrollo estaba enfocado en ejecutar los tests sobre *Travis CI*, mientras que en *CircleCI* construíamos la aplicación y aplicábamos *yarn* para analizar el código y su correcto formato.

3.2.4.3. Code Climate

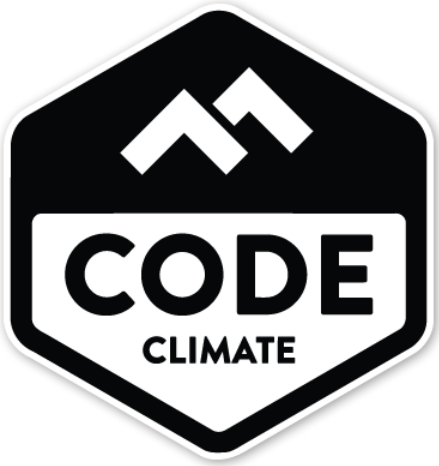


Ilustración 19. Logo Code Climate

Code Climate es un entorno cuya única finalidad es analizar la calidad del código y ejecutar los tests programados en su plataforma. Una de las características que le hacen increíblemente atractivo es que se pueden configurar los parámetros de evaluación (o también llamados controles de calidad), para amoldar esta herramienta a cada proyecto en concreto.

Esta herramienta establece unos controles de calidad por defecto, que pueden ser personalizados. Además, establece cinco niveles que establecen la calidad del código existente en el repositorio:

- **A**. Es la mejor calidad que se puede conseguir. Se obtiene con una cantidad menor a *2M*.
- **B**. Se obtiene con una cantidad comprendida entre *2M* y *4M.*
- **C**. Se obtiene con una cantidad comprendida entre *4M* y *8M.*
- **D**. Se obtiene con una cantidad comprendida entre *8M* y *16M*.
- **F**. Se obtiene con una cantidad mayor a *16M*.

Un *M* [9] es conocido como un punto de remediación (*remediation point*) que es una representación numérica de la cantidad de trabajo necesario para solventar un problema en el código.

3.2.5. 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.

3.2.5.1. ELK Stack

Versión 2.4.6

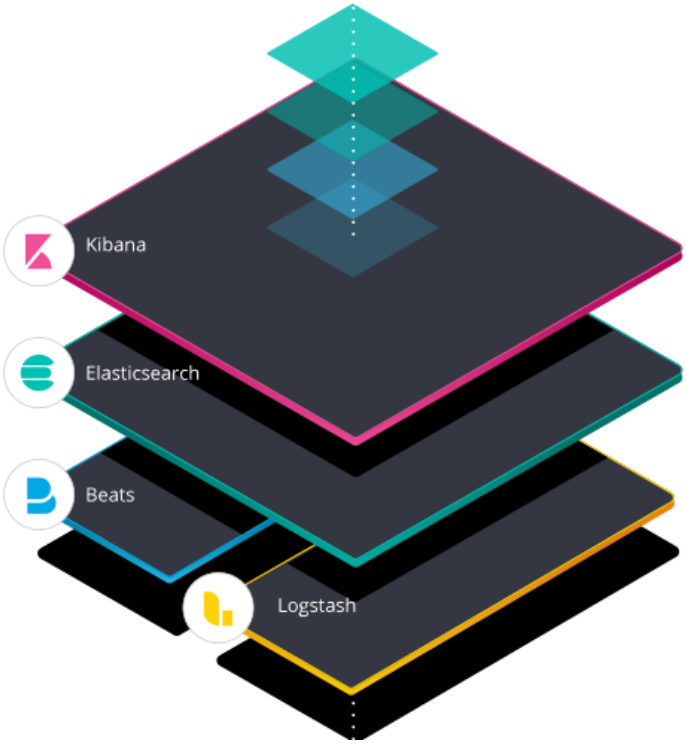


Ilustración 10. Logo ELK Stack

ELK Stack 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 *ElasticSearch*, 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 *LogStash* (o el cliente *Spring* que hemos desarrollado) y se visualiza y analiza con *Kibana*.

Ilustración 11. Logo ElasticSearch (izquierda) y Kibana (derecha)

Junto con las comparaciones presentes en el interfaz gráfico del proyecto, el usuario también tendrá acceso a *Kibana*, para visualizar y realizar otros tipos de análisis sobre la información almacenada.

3.2.6. 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.

3.2.6.1. Docker

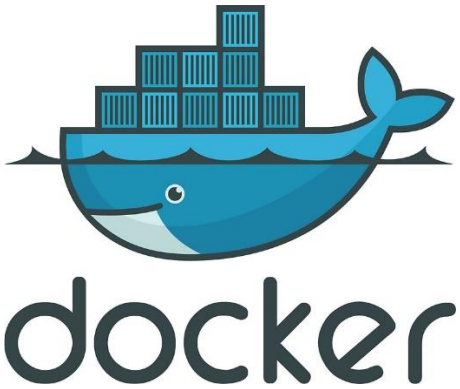


Ilustración 12. Logo Docker

Docker 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* [10] 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).

3.2.6.2. Docker Compose

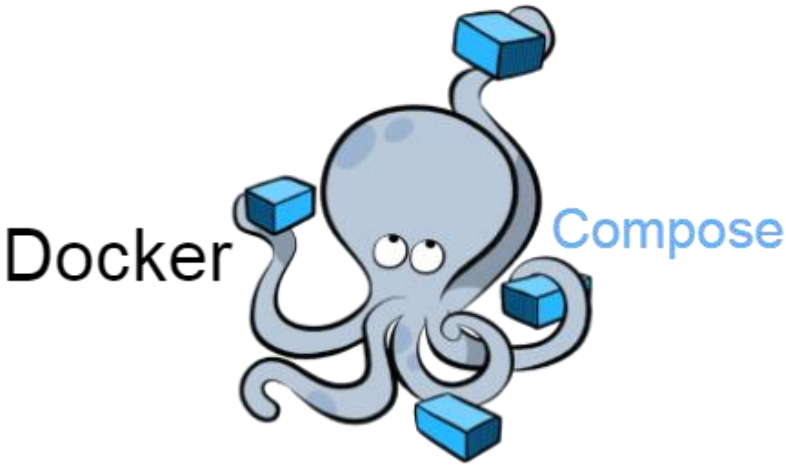


Ilustración 13. Logo Docker Compose

Docker Compose 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.*

3.2.7. Otras herramientas

Durante el desarrollo del proyecto, se han empleado otras herramientas como las descritas a continuación.

3.2.7.1. Postman

Ilustración 14. Logo Postman

Postman 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 nece-sidad 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.

3.2.7.2. Compodoc

Ilustración 15. Logo Compodoc

Compodoc 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*).

Referencias

[1] C. Jones, Applied Software Measurement, New York: McGraw-Hill, 1996.

[2] D. Pogue, «5 Most Embarrassing Software Bugs in History,» Scientific American, New York, 2014.

[3] TypeScript, «TypeScript - JavaScript that scales.,» [En línea]. Available: https://www.typescriptlang.org/. [Último acceso: 22 Abril 2018].

[4] AltexSoft, «The Good and the Bad of Angular Development,» 29 Septiembre 2017. [En línea]. Available: https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-angular-development/. [Último acceso: 20 Abril 2018].

[5] Wikipedia, «Node.js,» Wikipedia, 20 Abril 2018. [En línea]. Available: https://es.wikipedia.org/wiki/Node.js. [Último acceso: 22 Abril 2018].

[6] Wikipedia, «Objeto (programación),» Wikipedia, 20 Marzo 2018. [En línea]. Available: https://es.wikipedia.org/wiki/Objeto_(programaci%C3%B3n). [Último acceso: 3 Mayo 2018].

[7] Wikipedia, «Dependencias de software,» Wikipedia, 3 Septiembre 2014. [En línea]. Available: https://es.wikipedia.org/wiki/Dependencias_de_software. [Último acceso: 3 Mayo 2018].

[8] Wikipedia, «Maven,» Wikipedia, 13 Abril 2018. [En línea]. Available: https://es.wikipedia.org/wiki/Maven. [Último acceso: 4 Mayo 2018].

[9] Code Climate Docs, «Code Climate Glossary,» Code Climate Docs, 4 Octubre 2017. [En línea]. Available: https://docs.codeclimate.com/docs/code-climate-glossary. [Último acceso: 4 Mayo 2018].

[10] Wikipedia, «Contenedores de software,» Wikipedia, 31 Octubre 2017. [En línea]. Available: https://es.wikipedia.org/wiki/Contenedores_de_software. [Último acceso: 4 Mayo 2018].