Implementación de un framework para la inclusión de logros en ¡*Acepta el Reto!*

Trabajo de fin de grado
Grado en Ingeniería del Software
Facultad de Informática
Universidad Complutense de Madrid
Curso 2015-2016

Realizado por
Víctor Montaña Garrido

Proyecto dirigido por
Marco Antonio Gómez Martín
Pedro Pablo Gómez Martín

Agradecimientos

A mi familia y amigos, por el apoyo que me han dado durante toda la carrera, es por ellos que he llegado hasta aquí.

A mis compañeros de grado, que han compartido trinchera conmigo estos años.

A los profesores de la facultad de informática, por el esfuerzo que han hecho intentando enseñarnos y la paciencia que han demostrado intentando aguantarnos.

Al profesor Antonio Navarro, por enseñarnos, además de las asignaturas correspondientes, una actitud hacia el trabajo y verdadera ética profesional.

A mis directores de proyecto, sin cuya paciencia, comprensión y consejos no habría sido posible este trabajo de fin de grado.

A todos ellos, gracias.

Víctor.

Resumen

"Implementación de un framework para la inclusión de logros en ¡Acepta el Reto!" es un proyecto que tiene como objetivo la inclusión de un sistema de logros en ¡Acepta el reto! Los logros son reconocimientos, independientes de la funcionalidad de la web, dentro de ¡Acepta el reto!, que es una página web que propone ejercicios de programación y a su vez los compila y los ejecuta.

La labor del *framework* desarrollado es vigilar las acciones que los usuarios van realizando en el portal para otorgarles los distintos logros implementados. Durante el diseño se ha prestado especial atención a la posibilidad de añadir nuevos logros tras el despliegue del sistema y a la eficiencia en el proceso de comprobación de los mismos. Para demostrar su efectividad, se han hecho pruebas de carga utilizando los datos de usuarios, envíos y ejercicios actuales de la web.

Palabras clave

- Juez On-Line.
- Sistema de logros.
- Sistema de eventos.
- Ejercicios programación.
- Desarrollo de framework.
- Medallas sociales.

Abstract

"Implementation of a framework for the inclusion of achievements in Accept the Challenge!" is a project that aims to include a system of achievements in Accept the challenge! Achievements are acknowledgments independent to the functionality of the web, within Accept the challenge!, which is a website that proposes programming exercises and in turn compiles and executes them.

The mission of the framework developed is to monitor users' actions within the site in order to assign the implemented achievements. During the design special attention has been paid to the possibility of adding new achievements after system deployment, as well as efficiency in the process of checking them. To prove its effectiveness, load tests have been carried out using user data, assignments and exercises currently in the web.

Keywords

- Online Judge.
- Achievement system.
- Event system.
- Programming exercises.
- Framework development.
- Social medals.

Ningún mar en calma hizo experto a un marinero.
Anónimo

Índice

1. Introducción

La finalidad del siguiente trabajo de fin de grado es cerrar mis estudios en el Grado de Ingeniería del Software, donde se ha intentado, de la mejor manera posible, aplicar parte de los conocimientos aprendidos, durante los años de carrera, en un proyecto real y operativo.

Se ha intentado respetar aquellas metodologías de trabajo así como los conceptos básicos y avanzados de la programación de la mejor manera posible, de modo que no sólo importe el "Qué" se ha hecho, sino también el "Cómo", lo cual es, en mi opinión personal, la clave de una ingeniería.

Conceptos de arquitectura e ingeniería del software como son la cohesión, el acoplamiento, la escalabilidad, etc. han sido tenidos en cuenta durante toda la realización del trabajo y tratados con especial cuidado para mantenernos siempre dentro de los límites no solo necesarios si no, también, deseables.

También se ha tenido especial cuidado, además de porque era necesidad funcional, con la eficiencia de los algoritmos tanto en tiempo como en memoria, controlando en todo momento el consumo de estos y haciendo y rehaciendo el código hasta obtener una versión óptima de cada uno de los algoritmos.

1.1 Motivación

¡Acepta el reto! es un juez online, un repositorio de ejercicios de programación con la capacidad, a su vez, de compilar y ejecutar la solución propuesta por los usuarios a dichos ejercicios.

Establecida esa funcionalidad, se desea hacer más hincapié en la sociabilidad de la web, pues se ha visto que las interacciones entre usuarios están muy limitadas.

De entre las opciones posibles para iniciar un camino a una web más social, se deseaba implementar un sistema que premie a los usuarios por determinados méritos o hazañas, independientes de la propia resolución de los ejercicios: un sistema de logros.

1.2 Objetivos

El objetivo de este trabajo de fin de grado ha sido el de desarrollar un *framework* que posibilite la inclusión de logros dentro de ¡Acepta el reto!

Dicho *framework* deberá integrarse en el *backend* del servidor y tener la capacidad de, automáticamente, asignar logros a los usuarios cuando se cumplan determinadas condiciones.

Además el *framework* debe ser transparente para los usuarios y administradores, así como eficiente en el consumo de recursos, de modo que la inclusión del sistema sea factible en consideración de tiempo de ejecución y memoria consumida.

También será indispensable que el *framework* sea escalable, pues va a integrarse en "un organismo vivo" que irá evolucionando y creciendo, y además, aunque ahora se vaya a desarrollar un *framework* con algunos logros a modo de ejemplo, el sistema debe permitir, con un mínimo esfuerzo por parte de los administradores, la inclusión e integración automática de nuevos logros en el sistema.

Se considerará, además, condición necesaria la corrección de dicho sistema de logros, demostrando, mediante pruebas empíricas, que el funcionamiento es correcto y similar en cada una de las simulaciones, poniendo de manifiesto la satisfactibilidad de nuestro código.

Dicho *framework* quedará integrado en el *backend*, dejando como trabajo futuro su inclusión en el portal online.

1.3 Estructura de la memoria

En los capítulos siguientes podremos obtener información sobre el proyecto, teniendo así:

Un capítulo donde podremos obtener los antecedentes de los sistemas de logros y los jueces Online que es el capítulo 3. *Estado del arte*,

A continuación, en el capítulo 4. *Punto de partida*, relataremos brevemente el estado actual de la plataforma en la cual nos vamos a integrar: ¡Acepta el reto!

En el capítulo 5. *Primera aproximación* se relatará el estudio y la forma en la que se va a incluir el sistema de logros en la web ¡Acepta el reto!

A continuación tendremos dos capítulos: 6. *Implementación del sistema de eventos* y 7. *Implementación del sistema de logros*, durante los cuales relataremos los detalles técnicos de la implementación del proyecto.

Finalmente disponemos de un capitulo 8. *Aplicación del sistema de logros* en el cual se arrojarán datos sobre las simulaciones del sistema de logros y se estudiará, mediante estadísticas, el resultado de dicha aplicación.

Terminaremos con el capítulo 9. *Conclusiones*, en el cuál se analizará el resultado del trabajo así como las proposiciones a futuro que han ido surgiendo durante la realización del mismo.

2. Introduction

The purpose of this EOG project is to cap off my studies in the Bachelor of Engineering Software, and it has tried, as far as possible, to apply some of the knowledge acquired during the years of said career, to a real functioning project.

Efforts have been made to respect work methods as well as basic and advanced concepts, in such a way that not only "what" has been done is important, also the "how", which in my opinion is the key to engineering.

Concepts related to software architecture and engineering such as cohesion, coupling, scalability, etc. have been taken into account throughout the project and treated with special care to always remain within not only required but also desirable limits.

Special attention has also been given, in addition to the fact that it was functionally necessary, to the efficiency of algorithms both in time and memory, controlling at all times the consumption of these and doing and redoing the code to obtain an optimal version of each algorithm.

2.1 Motivation

Take the challenge! is an online judge, a repository of programming exercises with the ability, in turn, to compile and run the proposal by users to these exercises.

Once this functionality has been established, we wish to place more emphasis on the sociability of the web, as we have observed that interactions between users are very limited.
Among the possible options to pave the path to a more social web, we wanted to implement a system that rewards users for certain merits or achievements, independent from the actual solving of the exercises: an achievement system.

2.2 Goals

The aim of this EOG project has been to develop a framework that enables the inclusion of achievements within Take the challenge!
Such a framework should be integrated into the backend server and have the ability to automatically assign achievements to users when certain conditions are met.

Additionally, the framework should be transparent for users and administrators, as well as efficient in resource consumption, so that the inclusion of the system is feasible in terms of execution time and memory spent.

It will also be essential for the framework to be scalable, as it will be integrated into "a living organism" that will evolve and grow. Also, although we are presently developing a framework with some achievements as an example, the system should allow, with minimal effort by admins, to include and integrate new achievements in the system automatically.

Furthermore, it is deemed a necessary condition for the achievement system to be correct, demonstrating through empirical evidence, that the operation is correct and similar in each of the simulations, highlighting the satisfiability of our code.
This framework will be integrated into the backend, leaving the inclusion in the online portal as a future mission.

2.3 Memory Structure

In the following chapters we can obtain information about the project, as follows: A chapter where we can obtain the predecessors of achievement systems and online judges in Chapter 3. State of the art,

Then in chapter 4. Starting point, we shall briefly describe the current state of the platform on which we will integrate: Take the challenge!

In Chapter 5. First approach, we will narrate our study and method with which we will include the achievement system on the web Take the challenge!

Then we will have two chapters: 6. Implementation of system events and 7. Implementation of achievement system, during which we will describe the technical details of project implementation.

Finally we have chapter 8. Implementation of achievement system, in which we will receive data on achievement system simulations and the outcome of the application will be studied by statistics.

Ending with Chapter 9. Conclusions, in which the result of the project will be analyzed, as well as future proposals that have emerged during the implementation.

3. Estado del arte

3.1 Sistema de logros

*"En el lenguaje de los videojuegos, un **logro**, también conocido a veces como un **trofeo** o un **desafío**, es una meta definida fuera de los parámetros de un juego. A diferencia de los sistemas de misiones o niveles que suelen definir los objetivos de un videojuego y tienen un efecto directo en el juego, la gestión de los logros por lo general se lleva a cabo fuera de los confines del ambiente de juego."* Extracto de la Wikipedia [1]

La definición proporcionada por la Wikipedia muestra el hecho de que los logros no son inherentes al juego, sino añadidos que funcionan de manera paralela a él, alargando su vida útil, la rejugabilidad, etc. Ha sido, principalmente, por estas razones por las cuales los logros se han propagado con tanta velocidad, siendo a día de hoy parte fundamental de un videojuego.

Deberíamos partir de la base de por qué se originaron los sistemas de logros, pues, en un principio, no formaban parte de los videojuegos.

Las primeras apariciones de sistemas de logros datan de principios de los años noventa, aunque por aquel entonces eran considerados como objetivos *bonus*, definición a día de hoy desfasada.

Originalmente los videojuegos estaban pensados para ser jugados por una persona, en su ordenador y generalmente durante un tiempo definido, aunque no necesariamente ajustado, pero si acotado: "*El modo campaña de este juego está considerado para ser completado en cuarenta horas*" por ejemplo. Sin posibilidad de compartir información acerca de su experiencia o modo de juego (por medios telemáticos).

Y así fue durante mucho tiempo, hasta que hizo aparición internet, cambiando para siempre los videojuegos. Aunque los sistemas de logros, como tal, son anteriores a Internet, fue a consecuencia de este cuando se popularizaron, evolucionaron, se desarrollaron y se hicieron condición *sine qua non* de los videojuegos. Con internet llegó la socialización de los videojuegos, ahora había todo un mundo con el que jugar, interactuar, compartir, no sólo consistente en jugar con otras personas, sino en poder competir con ellos, bien sea de forma directa o mediante puntuaciones (degenerado, posteriormente, en logros).

De este modo, surgieron los logros como la conocemos hoy día, como una forma de representar las medallas, hazañas, méritos, etc. que hemos podido obtener por la forma en que jugamos o quizá algo específico que hayamos conseguido y que sea válido destacar.

Después surgió otra modalidad de juego, con la inclusión de los juegos *Sandbox* [2] que tienen la característica de no ser juegos lineales y no tener un principio y un final al uso, si no que la forma de jugarlo es dependiente del usuario. Estos juegos establecen un mundo y una línea de juego, que puede ser seguida o no, a elección del usuario, dando una libertad total de acción y obra. Esto aceleró la inclusión del sistema de logros, pues obtener determinadas metas personales, aun cuando estas no están premiadas dentro del juego, requería algún tipo de compensación o publicación. De hecho, el sistema de logros es tan notable que a día de hoy es difícil imaginar un juego de primera categoría que carezca de ellos. Algunas plataformas exigen a los desarrolladores que los incluyan.

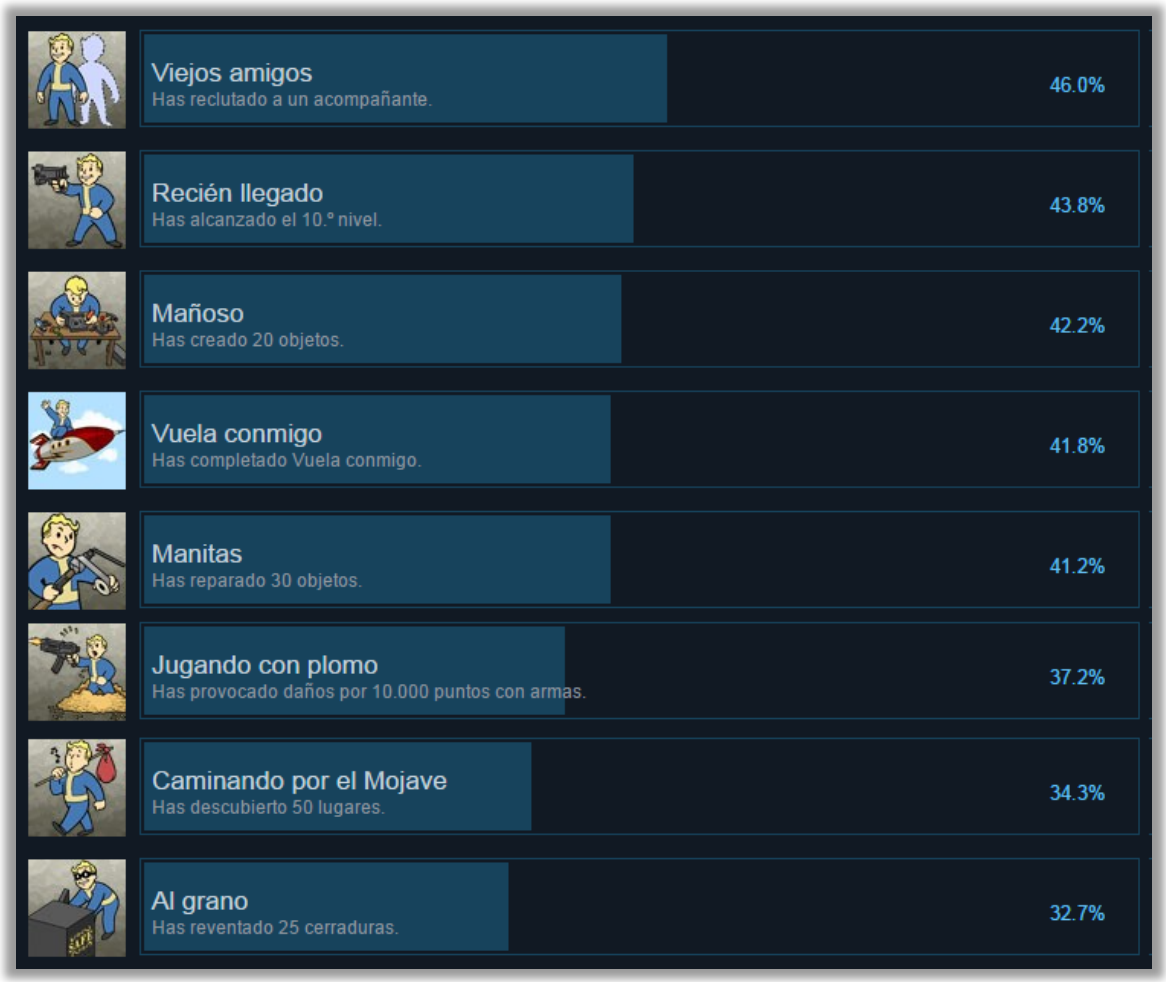
Todo lo anterior nos condujo a un sistema de logros, cuyas metas pueden ser tan variopintas como:

- Cazar un determinado número de palomas distribuidas por diferentes ciudades. Logro clásico de la saga Assassins Creed [3].
- Obtener una cantidad de diamantes ocultos por el mapa. Logro clásico de la saga Far Cry [4].
- Robar un coche en particular. Logro típico en cualquiera de los Grand Theft Auto [5]
- Descubrir algún lugar único y secreto del juego. Saga The Elder Scrolls [6]

Los logros se han hecho tan populares e indispensables que de hecho han sido incluidos de forma directa en la mayoría de plataformas de videojuegos como pueden ser Steam [7] u Origin [8], PlayStation Network [9], Google Play [10], etc.

Estas plataformas nos permiten, además, obtener datos sobre nuestros logros, los de nuestros amigos, compararlos y además saber cuánta gente tiene dicho logro o cual es el porcentaje de obtención general del logro a nivel mundial.

Detalle de los logros del juego *Dawn of war 2*



Porcentaje de obtención de los logros a nivel mundial, juego *Fallout: New Vegas*.

La existencia de los logros en un videojuego tiene, además, un efecto colateral y es alargar la vida útil de los videojuegos, al proponer al usuario jugarlos de nuevo con distintas estrategias en busca de los logros que no haya podido obtener, etc.

Existen videojuegos que tienen los logros ocultos y sólo los conoces una vez que los desbloqueas.

Con el tiempo los logros saltaron directamente a otros ámbitos de internet, entre ellos creo que es necesario mencionar los foros, como StackOverflow [11], donde se premia mediante pseudo-títulos y medallas la participación de los usuarios, incluso pueden proporcionar permisos especiales dentro de la web.

3.2 Jueces en línea

Los jueces en línea son páginas web que contienen una serie de problemas de programación. A diferencia de un repositorio de problemas, estos jueces online tienen la capacidad de recibir un código fuente, compilarlo y ejecutarlo. Después le proporcionan unos datos de entrada, capturan los datos de salida y, con ellos, el tiempo consumido, la memoria ocupada, etc. Finalmente proporcionan un veredicto sobre el código enviado por el usuario.

Así, sería el equivalente a un profesor que propone ejercicios y luego te los corrige, pero funcionando en la plataforma más masiva existente, Internet.

Hay multitud de jueces y todos se ajustan a los principios básicos contados con anterioridad, aunque algunos sean más completos que otros. Por ejemplo es habitual que cada juez se limite a unos lenguajes específicos, siendo los más comunes, como es lógico, C++ [12] y Java [13], aunque los hay que acepten muchos más.

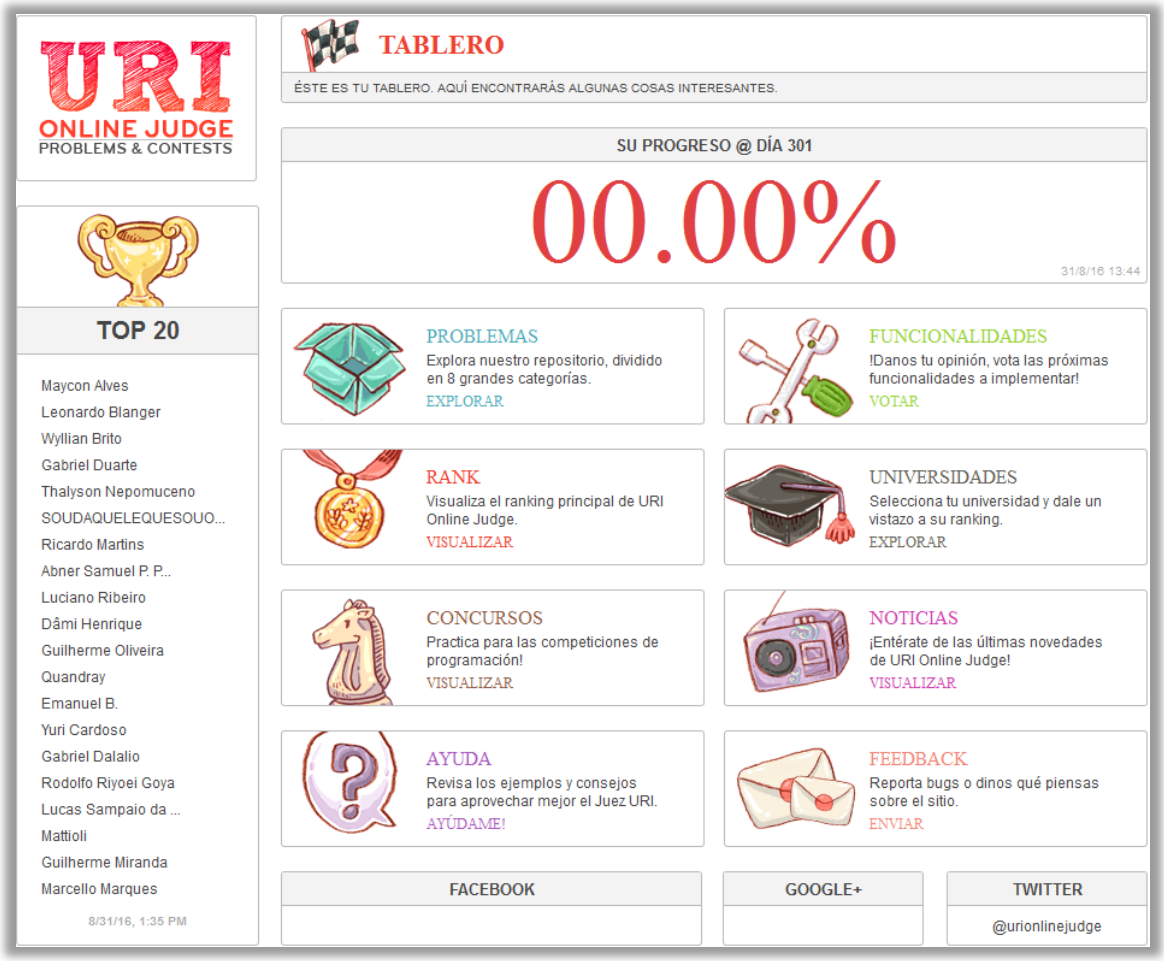
También varían mucho en cuanto a la forma de interactuar de los usuarios dentro de la web, yendo desde lo más básico, usuarios independientes con su login y ninguna interacción entre ellos a lo contrario, es decir, prácticamente una red social donde los usuarios tienen perfiles, logros, estadísticas, rankings, etc.

Es importante recalcar que algunos de estos jueces online, como por ejemplo TopCoder [14] tienen una importante reputación y de hecho Google [15] los recomienda como base para preparar una entrevista con ellos.

También debemos añadir que aunque pueden ser utilizados a modo de competición o en las propias competiciones, la finalidad general de estos jueces es la del autoaprendizaje, ponerte a prueba a ti mismo, ver hasta dónde y cómo eres capaz de llegar, etc. Es decir, se podría asignar un valor didáctico, en general, como finalidad de los mismos.

En las páginas siguientes veremos, por encima, alguno de estos jueces online, que servirán de ejemplo.

URI Online Judger [16]



Es un juez online muy enfocado a la sociabilidad, pues contiene multitud de maneras de interactuar con los usuarios, incluidos perfiles de usuario, opción de envío de problemas, sistema de rankings para ver quien ha resuelto en menos tiempo, quien tiene más puntos, etc.

Contiene a su vez un sistema de logros ya integrado y operativo, bastante completo.

Este juez dispone así mismo de un gran repositorio de problemas, ordenado por categorías, para poder hacer ejercicios específicos de la rama que más te interese.

Admite envíos en: C, C#, C++, Java, Python y Ruby.

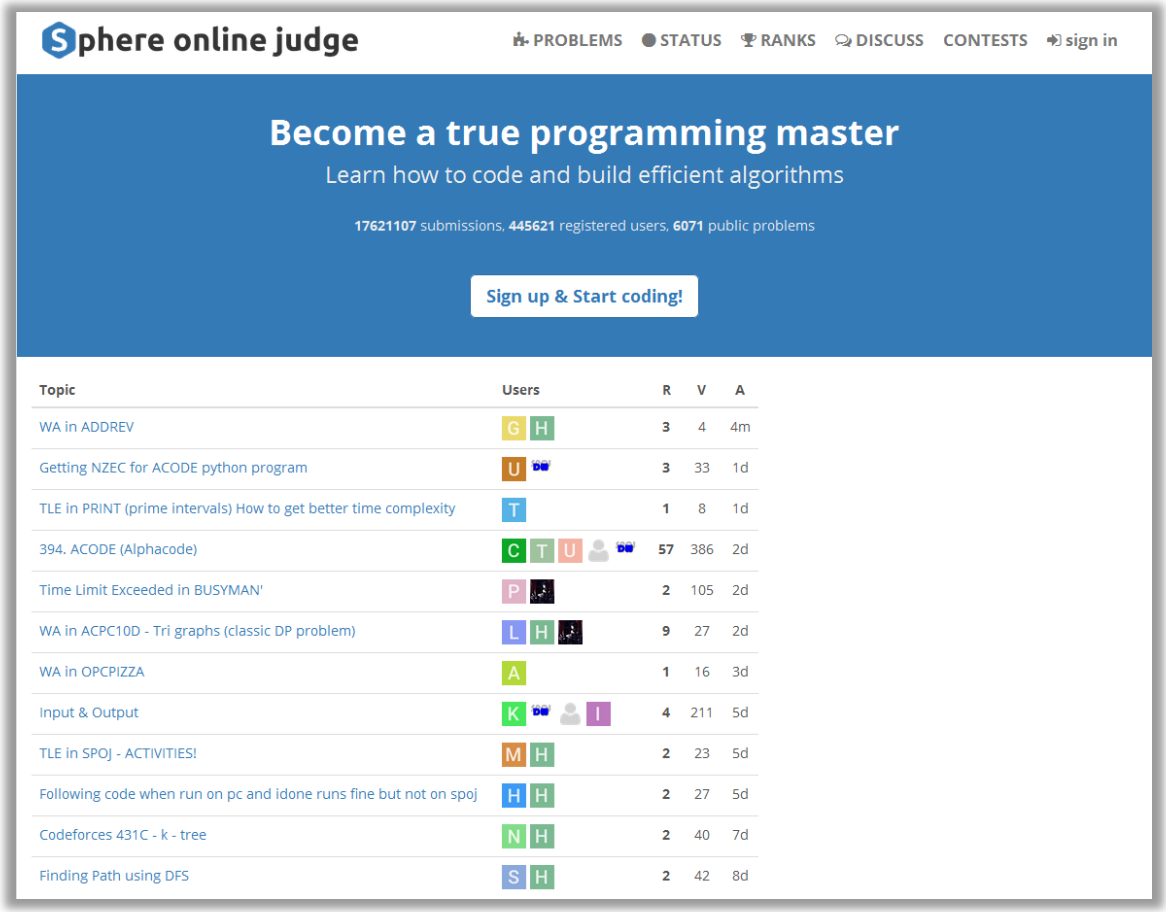
UVa Online Judge [17]

Es un juez online que ha sentado las bases, pues es muy antiguo, tiene un catálogo de problemas bastante estructurado.

Dispone de estadísticas dentro de los propios problemas individuales lo cual sirve de mucha ayuda a la hora de seleccionarlos, pues puedes ver los porcentajes de acierto, etc.

Así mismo, para cada usuario, se dispone de estadísticas sobre los problemas que se han intentado resolver, siendo esta información pública, de modo que nos permite ver que usuarios revuelven los problemas en un primer intento, quienes los resuelven a base de intentos, etc.

Sphere Online Judge [18]



Este juez online dispone de una estética muy limpia, con una página web muy poco cargada que permite una navegación muy cómoda. Está muy orientado a la utilización de foros y la sociabilidad de los problemas, lo que da lugar a correcciones de ejercicios entre varias personas dentro de los foros.

Dispone de un gran repositorio de ejercicios muy bien categorizados, permitiendo a los usuarios elegir un problema muy específico dentro del ámbito que deseen.

Esta página web se caracteriza por la cantidad de estadísticas que ofrece sobre prácticamente todo, muestra estadísticas sobre los usuarios y sus envíos, sobre la resolución de determinados ejercicios, etc.

Además ofrece mucha información sobre las organizaciones a las que pertenecen los usuarios, países, etc.

4. Punto de partida

El portal de internet ¡Acepta el reto! [19] es un juez online, donde se proponen ejercicios de programación que pueden ser resueltos en varios lenguajes. Dicha plataforma, al ser un juez online, tiene la característica de que no sólo propone ejercicios de programación, sino que además, tiene la capacidad de corregirlos automáticamente y notificar a los usuarios el resultado de su ejercicio, proporcionando además, en el caso de error, información adicional sobre porque ha fallado el problema, sea por límite de tiempo, errores de compilación, etc.

Página principal de ¡Acepta el reto!

Nace como una iniciativa de dos profesores de la Facultad de informática de la Universidad Complutense de Madrid, directores, además, de este trabajo de fin de grado. En 2011 participaron en la puesta en marcha del proyecto ProgramaMe, concurso de programación para alumnos de Ciclos Formativos de Formación Profesional y que fue el origen de la idea. Con esa base de ejercicios de programación y con la idea de ampliarlo y darle difusión, en el año 2013 se comenzó con la implementación del backend por parte de los profesores, delegando la parte del *frontend* a estudiantes, que generaron dos versiones preliminares, que sirvieron de base para pruebas.

Finalmente en Febrero de 2014 se pone en marcha ¡Acepta el reto! con el primer envío a la misma, alcanzando los 65.535 envíos en marzo del 2016.

Podemos interactuar con la web, leer los problemas, etc. pero es necesario estar registrado para resolver los problemas y subirlos, delegando en la página las tareas de corrección y puntuación.

El proyecto *"Implementación de un framework para la inclusión de logros en ¡Acepta el Reto!"* forma parte de una serie de mejoras que quieren incluirse en Acepta el reto con la finalidad de potenciar la parte social de la página, pues al no pertenecer al *core* de la página, hasta ahora han sido dejados en un segundo plano, permitiendo sólo algunas opciones de personalización para los usuarios, como introducir su foto, datos personales, etc. pero limitando en extremo las interacciones entre ellos.

Actualmente, septiembre del 2016, el portal cuenta con más de 3.000 usuarios registrados, más de 250 problemas y cerca de 80.000 envíos por parte de los usuarios, es un sistema que se encuentra en un estado de funcionamiento óptimo.

Los problemas propuestos son aplicaciones de consola, que recibirán datos por entrada estándar y enviarán unos resultados por salida estándar. Los problemas disponibles son de distinta índole: programación, concursos, etc. Además hay una organización secundaria en la cual podremos filtrar por estructuras de datos, algoritmos, etc.

Los ejercicios enviados serán almacenados para, posteriormente, ser evaluados. Las soluciones a los ejercicios enviados por los usuarios serán sólo visibles para ellos, considerándose privado el código de cada usuario.

Cuando se recibe un problema, el juez ejecuta sobre él una serie de casos de prueba secretos, observando que la salida sea correcta o no. Dependiendo del resultado de dicha evaluación, se proporcionará un veredicto. Las opciones disponibles son:

- Aceptado (AC): El problema es correcto, en ejecución, en tiempo y en salida esperada.
- Error de presentación (PE): Hay un error en el formato de los datos de salida del ejercicio. Por lo demás el ejercicio estaría correcto.
- Respuesta Incorrecta (WA): El resultado del ejercicio es incorrecto.
- Error de compilación (CE): El código enviado tiene errores de sintaxis y no ha sido posible compilarlo, ni probar los casos de prueba.
- Error de ejecución (RTE): Se pudo compilar y ejecutar el programa, pero la ejecución fue interrumpida de manera inesperada (excepciones, etc.)
- Tiempo límite superado (TLE): El ejercicio ha superado el tiempo máximo permitido y por tanto la ejecución se ha interrumpido.
- Límite de memoria excedido (MLE): El ejercicio ha superado el espacio en memoria permitido y la ejecución se ha interrumpido.
- Límite de salida superado (OLE): La salida proporcionada por el ejercicio es superior al límite permitido, el ejercicio es, por tanto, incorrecto.
- Función restringida (RF): EL juez ha interrumpido la ejecución debido a que ha encontrado alguna función inadecuada o peligrosa para el servidor.
- En cola (IQ): El ejercicio aún se encuentra pendiente de corrección.
- Error interno (IE): El servidor ha tenido algún problema ajeno al ejercicio o está en mantenimiento, deberá enviarse de nuevo el código para una futura corrección.

4.1 Estructura de Acepta el Reto

La arquitectura de ¡Acepta el reto! está dividida en varias capas. A continuación se describirá el funcionamiento de la misma de la forma más general posible, sin entrar en detalles técnicos, que son, para nuestro objetivo, irrelevantes.

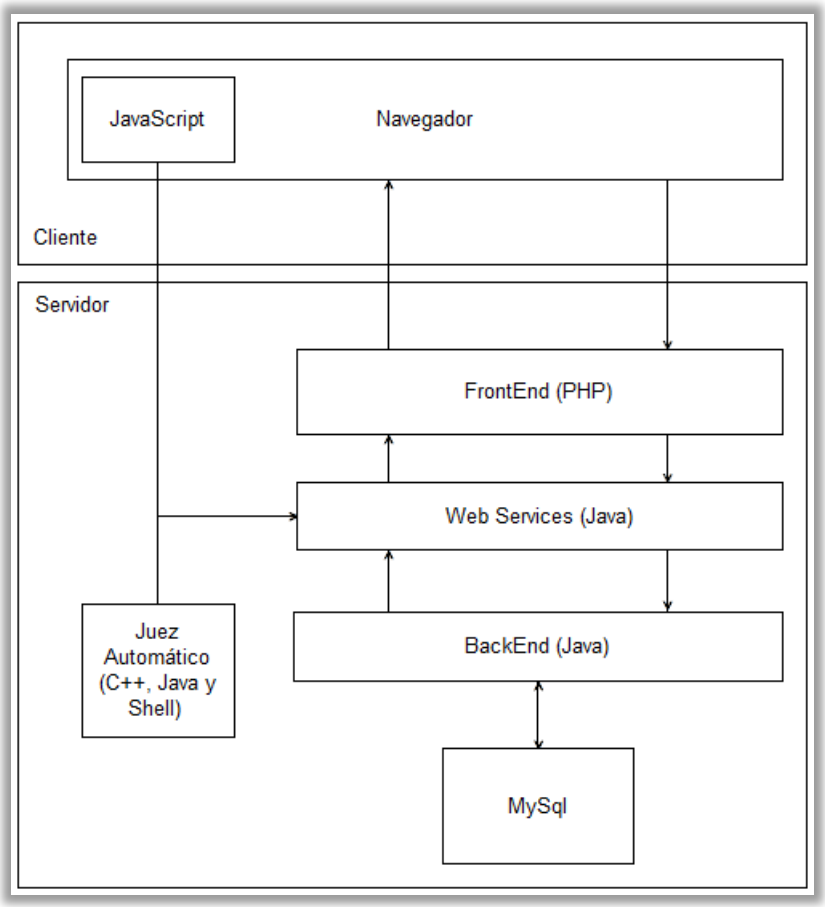
Cuando el usuario entra a la página se conecta a un *frontend*, dicho *frontend* está implementado con HTML [20] + JavaScript [21] y PHP [22]. Así, la página mostrada a los usuarios es montada por la combinación de ambos (PHP y JavaScript) que acceden a los servicios web.

Los servicios web se integran en una parte intermedia del sistema, y permiten la interacción entre el frontend y el backend.

El backend está implementado en Java [23], utilizando Hibernate [24] y MySql [25], integrado en un sistema Apache [26] + Tomcat [27] y es quien lleva la mayor parte de la carga de trabajo, pues es la encargada de mantener al resto del site, así como corregir los ejercicios, etc.

El alcance de este trabajo de fin de grado es integrarse en el *backend*, incorporando funcionalidad que permita la integración de un sistema de logros, siendo este uno de los inicios de la socialización de la web, aunque no el único, pues dicha web ya dispone de un sistema de rankings.

Así pues, ha sido necesario analizar el funcionamiento de la web y estudiar la mejor forma de implementar el sistema de logros. Buscando que no sea sólo eficiente, sino que además tenga carácter retroactivo. Siendo este último punto uno de los hitos del trabajo, pues deberá tener en cuenta no sólo las futuras interacciones de los usuarios con la web, sino además la base instalada que consta desde la puesta en marcha en 2014.



Esquema simplificado de la estructura de ¡Acepta el reto!

Otro de los puntos críticos del trabajo ha sido el de integrarse totalmente en el sistema. Con el objetivo de hacer el proceso lo menos intrusivo posible, parasitando el sistema y funcionando en paralelo a él, sin necesidad de modificar su funcionalidad, sino incorporándose a ella de forma transparente y automática. Era condición explícita del trabajo que el sistema de logros no requiera de ninguna interacción humana.

También ha sido clave para el trabajo la atomicidad del sistema de logros, evitando cualquier tipo de acoplamiento con el resto de la aplicación. El sistema de logros puede ser retirado completamente de la aplicación sin necesidad de modificar el código de la misma.

Al comienzo del proyecto se disponía de dos proyectos de java, uno que implementaba las clases y el funcionamiento interno (ACR-MODEL) y otro que implementaba las ejecuciones, la conexión a base de datos, etc. (ACR-MODEL-TOOLS).

Se ha dispuesto, a su vez, tanto a nivel de java como de base de datos, de todas aquellas clases relacionadas con usuarios, envíos, problemas, etc. que eran necesarias para tener una base estable sobre la que trabajar, y que, de hecho, forman parte real de Acepta el reto, el proyecto donde nos íbamos a integrar.

Para la realización de este trabajo ha sido necesario extender la base de datos, donde se almacenan los logros, los eventos y ciertas interacciones con el sistema establecido y también ampliar la parte java del backend, especialmente la parte de ACR-MODEL, para la creación de las clases que pertenecen a los sistemas de logros y eventos, así como para la implementación de los propios logros y también al ACR-MODEL-TOOLS donde se han implementado las aplicaciones que nos permiten ejecutar y demostrar que el sistema está, efectivamente, funcionando.

5. Primera aproximación

Para la incorporación del sistema de logros ha sido necesario un estudio previo del sistema, analizando las opciones posibles, el esfuerzo requerido, la viabilidad, consumo de recursos, etc.

Partimos de la base de un sistema en ejecución y con multitud (miles) de interacciones ya realizadas. Un sistema con miles de usuarios y cientos de problemas: un sistema vivo y en funcionamiento.

Una posible implementación del sistema de logros (caso 1), clásica e ineficiente, es aquella que cada cierto tiempo recorre la base de datos, comprobando cada uno de los logros, y los asigna cuando corresponda desde cero.

Este caso nos propone una serie de problemas de eficiencia, pues implicaría una sobrecarga para el sistema que desperdiciaría recursos de forma indiscriminada. Esto supone una muy grave incidencia, pues dichos recursos son necesarios para el funcionamiento del core de la web, es decir, la corrección y mantenimiento del sistema de envío de ejercicios.

Un sistema que cada muy poco tiempo requiera un repaso de toda la base de datos, miles de comprobaciones, sobrecarga del procesador, para sólo asignar, en aquellos casos que procedan, un par de logros es un sistema totalmente inviable.

Además, como posteriormente se demostrará [1], este funcionamiento resulta inasumible, pues una ejecución sobre toda la base instalada requiere en torno a una hora de ejecución, lo cual imposibilita las llamadas al demonio, que estaban calculadas en torno a una llamada cada diez minutos.

Además este es un problema que irá en incremento, porque a medida que aumente el número de logros insertados en el sistema, para cada aplicación, sobre toda la base instalada, se requerirá más tiempo y ese tiempo será requerido en todas y cada una de las ejecuciones del sistema de logros, haciéndolo totalmente inmanejable.

También existe otro modelo de funcionamiento (caso 2) que resulta todavía más ineficaz, consistente en repasar toda la base de datos cada vez que llegara un

[1] Ver apartado Aplicación del sistema de logros

envío, pero esto es tan ineficiente que fue descartado en los primeros estados del proyecto. Además este segundo caso limita bastante la funcionalidad, por ejemplo la capacidad de otorgar logros por haberse conectado ciertos días, etc.

En los anteriores casos, para un logro como puede ser ***Siguiendo al conejo blanco***, que comprueba si un usuario tiene diez ejercicios AC (correctos), una posible implementación sería cargar en memoria, desde la base de datos, bien cada cierto tiempo (caso 1), bien cada vez que llega un envío (caso 2), todos los usuarios existentes en el proyecto (3500) y se comprueba, para cada uno de ellos, mediante consultas SQL, cuantos logros tiene como aceptados. Si el resultado es mayor o igual a diez, se asigna el logro.

El comportamiento anteriormente descrito puede ser factible una sola vez, para la inclusión del logro, pero resulta inasumible para los casos 1 y 2.

Se llegó entonces a la necesidad de optimizar este proceso, haciendo que el sistema sólo tenga que comprobar los nuevos sucesos que hayan ocurrido en el sistema desde la última vez que se comprobó, es decir, surgió la necesidad de llevar un registro de los sucesos que fueran teniendo lugar en el sistema. Esto puso de manifiesto la necesidad de un sistema de eventos.

El sistema de eventos es un sistema capaz de registrar todas las interacciones de la web, almacenándolas para su futuro análisis. Este sistema también ha tenido en cuenta la mayoría de las necesidades funcionales del sistema de logros, es decir, transparente para el usuario, transparente para el sistema, actuando automáticamente a modo de parásito, integrado en aquellas clases que provoquen eventos y escalable. Tal y como se ha definido e integrado el sistema de eventos es posible añadir nuevos eventos con sólo definirlos y añadirlos en la clase y método de java donde pueda ser necesario.

Tenemos pues, un sistema de eventos que nos permite comprobar los últimos hechos ocurridos en el sistema y, en base a ellos, actuar. Esto permite al sistema de logros hacer pequeñas y rápidas comprobaciones sólo con aquellos usuarios, aquellos problemas, aquellas interacciones, etc. que sean recientes, que sean realmente susceptibles de conllevar un logro, limitando de carga de trabajo al servidor y evitando la mayor parte de trabajo inútil, conllevando, además, apenas unos segundos de cálculo.

Dicho sistema de eventos nos permite asignar los nuevos logros que sean susceptibles de otorgarse como consecuencia del ciclo de vida de la aplicación,

pero nos plantea un problema con la base instalada, lo cual era una de las condiciones requisito de este trabajo de fin de grado.

La solución decidida para solventar esta incidencia es la de diferenciar dos posibles situaciones excluyentes entre sí: un logro es nuevo o un logro es viejo.

Un logro nuevo es aquel que acaba de incluirse en el sistema, bien sea con la puesta en marcha del sistema de logros (todos los logros serán nuevos) bien sea con la incorporación de nuevos logros en el futuro.

Un logro viejo es aquel que ya estaba incluido en el sistema y forma parte de él, con usuarios que poseen dicho logro y comprobaciones del mismo ya realizadas.

El sistema de asignación de logros es capaz de identificar estas dos situaciones, para cada logro, y actuar de forma diferente en cada uno de ellos.

La principal diferencia en las formas de actuación es que los logros nuevos requieren de una "aplicación" en el sistema, teniendo, esta vez sí, que realizar un barrido de toda la base de datos, comprobando y asignando cuando sea necesario. La novedad es que este proceso sólo tiene que realizarse una vez, dado que después el nuevo logro habrá sido absorbido por el sistema y no volverá a ser necesario dicho proceso.

Los logros viejos, ya integrados en la aplicación, sólo requerirán la "actualización", comprobando las interacciones que puedan provocar la asignación de un logro, conforme vayan llegando, permitiendo a cada logro, dentro de su código, identificar qué tipo de eventos le pueden afectar. De este modo, un logro correspondiente las horas de conexión del usuario, sólo obtendrá información de los eventos que le afecten (Eventos de login), descartando el resto y siempre desde la última vez que fue comprobado, es decir, el último evento que tenga registrado ese logro.

Volviendo al logro que teníamos como ejemplo, ***Siguiendo al conejo blanco*****,** una actualización, mediante el sistema de eventos, supondría, comprobar la última vez que el logro fue analizado y, si hay eventos recientes, comprobar cuáles de estos eventos son susceptibles de asignar el logro, que serían los de envíos, de esos eventos, extraemos los usuarios y para cada uno de esos usuarios, en caso de que no posean ya el logro, comprobaríamos en base de datos cuales tienen diez ejercicios aceptados.

Dado que la cantidad de eventos por comprobación variará cada vez, no se pueden dar cifras exactas, pero a día de hoy estaríamos hablando de unas 5-8 comprobaciones, en lugar de las 3500 que supone el repaso a base de datos.

Podemos afirmar, por lo tanto, que el sistema de eventos agiliza la comprobación y asignación, volviéndola ligera y eficiente.

6. Implementación del sistema de eventos

Como se vio en el capítulo anterior, es preciso contar con un sistema de eventos, en ¡Acepta el reto! no existía y ha sido necesario implementarlo, no sólo para la siguiente implementación del sistema de logros, sino como base para otro tipo de implementaciones que puedan venir en el futuro y depender de él.

Actualmente tenemos tres tipos de eventos: eventos de acceso, eventos de envíos y eventos de logros.

Los eventos de acceso se originarán cuando un usuario entre al sistema, cuando un usuario se loguee en el sistema, etc. Estos sucesos crearán un evento que será almacenado por si se desea tratar. Para su inclusión ha sido necesario añadir una llamada de creación de evento, con los parámetros correspondientes, dentro del método que guarda los usuarios en base de datos.

Los eventos de envíos se darán cuando un usuario realice un envío de un ejercicio, cuando un envío haya sido corregido, etc. Para su inclusión en el sistema ha sido necesario modificar el método de persistencia de envíos, añadiendo una llamada de creación de evento con los parámetros que sean necesarios.

Los eventos de logros se generarán cuando un usuario reciba un logro. En el método de asignación de logros se ha incluido una llamada a la creación de un evento del tipo logro.

Cada uno de estos eventos, tendrá a su vez un subtipo, que será quien aporte información específica del evento. Así, para el evento de envío, será el subtipo quien identifique si es un envío aceptado, si es un envío evaluado, etc.

La finalidad de esta clasificación por tipos y subtipos consiste en que los eventos tienen unos campos dinámicos (cuatro) que llevan información adicional del evento. Dicha información varía en función del tipo y el subtipo. Se hizo esta clasificación para poder permitir la variabilidad en esa información sin necesidad de tener un número indeterminado de columnas en la tabla, obteniendo una mejora en el espacio y los recursos consumidos, dado que en caso contrario estarían, la mayoría de columnas, vacías salvo las correspondientes al evento.

Así por ejemplo, en un evento de envío, el primer campo variable contendrá el identificador del usuario, el segundo contendrá el problema resuelto (o intentado), etc. Mientras que en un evento de logro el primer campo se

corresponde con el identificador univoco del usuario y el segundo campo se corresponde con el logro asignado.

Dichos tipos de eventos están definidos en un enumerado (será comentado después) y los subtipos están definidos como constantes, para poder ser utilizados con lenguaje natural dentro del código, facilitando así la reutilización del código y podrán crearse tantos como se desee o sea necesario.

Se tienen, además, otra serie de campos tales como un texto en lenguaje natural, que ofrece información sobre el evento que podría ser utilizada, fácilmente, en un futuro, para la creación de un sistema de log. Así como una marca de tiempo, para saber la hora en que se generó el envío.

Los eventos se insertan en la base de datos como si se tratara de una cola, pues todos llevan un identificador único, que se va incrementando de manera automática, además de una marca de tiempo, permitiéndonos así trabajar con ellos tanto usándola en modo *First in-first out* como en modo *Last in-first out*.

De hecho, este punto ha sido clave, pues el sistema de eventos nos permite obtener los eventos a partir de un determinado momento, bien sea mediante identificador o mediante la marca de tiempo, obteniendo todos los posteriores.

Además, por las características de almacenamiento de los eventos, el sistema permite la escalabilidad, pues basta con crear un nuevo símbolo del enumerado para cada tipo de evento y este irá con su identificador de evento en la base de datos. Después sólo es necesario incluir una llamada de creación de evento dentro de la clase que queramos que sea desencadenadora de ese evento. Así pues, los posibles usos del sistema de eventos son indeterminados, pero se ha abierto un abanico de opciones muy útil de cara a futuras implementaciones.

El sistema de eventos consta de dos clases: **Events.java** y **EventsDAO.java**, así como de un enumerado **EventType.java**.

Para la inclusión del sistema ha sido necesaria también la modificación de la base de datos, añadiendo una nueva tabla con los campos necesarios:

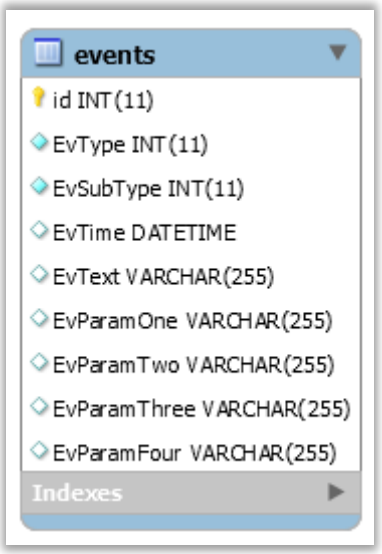


Tabla de eventos

Events.java

Es la clase que parametriza la entidad de los eventos (de cara a Hibernate y la relación con la base de datos). Nos permite hacer instancias de los eventos.

Cada evento consta de los siguientes atributos, debidamente parametrizados en la base de datos:

- **Id**, auto-incremental, será el identificador unívoco del evento.
- **EvType**, entero. Nos indica el tipo del evento.
- **EvSubType,** entero. Nos indica el subtipo del evento.
- **EvTime**, date. Marca de tiempo de creación del evento.
- **EvText,** cadena. Información en lenguaje natural del evento.
- **EvParamOne,** cadena, información adicional.
- **EvParamTwo,** cadena, información adicional.
- **EvParamThree,** cadena, información adicional.
- **EvParamFour,** cadena, información adicional.

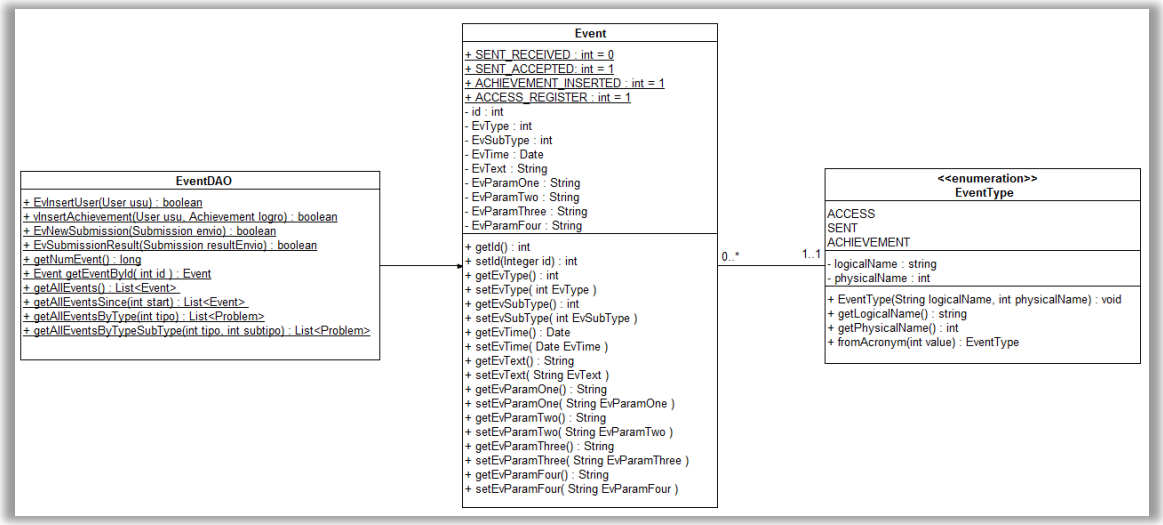


Diagrama de clases de los eventos

EventsDAO.java

Es la clase que nos permite el acceso a la tabla de eventos. Coordinado con Hibernate nos permite, muy fácilmente, trabajar con los registros en base de datos, manipularlos, realizar nuevas inserciones, etc.

Además esta metodología de trabajo nos permite diferenciar la lógica del proceso de los accesos a base de datos, mejorando la encapsulación, así como la posibilidad de migración a otro sistema de gestión de datos.

También nos permite la fácil inclusión (y por ello mejora la escalabilidad) de nuevas consultas en base de datos directamente desde el código java.

EventsType.java

Esta es la clase que define los enumerados para los tipos de eventos, tiene los tres tipos de eventos comentados con anterioridad: de envío, de logros y de acceso.

Nos permite trabajar con estos datos en un lenguaje natural, aunque luego, a la hora de trabajar con la base de datos, sean convertidos a los números correspondientes de identificación.

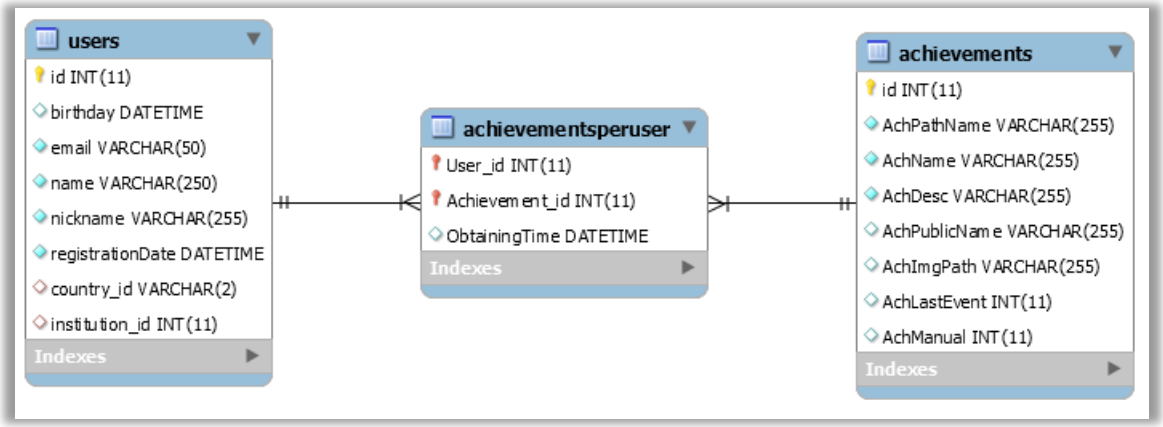
Contiene, así mismo, los métodos necesarios para trabajar con los enumerados.

7.Implementación del sistema de logros

Como se ha comentado, es indispensable la generación de un *framework* que funcione de forma transparente y automática, de modo que asigne los logros a los usuarios cuando corresponda, sin un consumo excesivo de recursos (bien tiempo, bien memoria) y que sea escalable, permitiendo ampliar el número de logros sin necesidad de modificar dicho *framework* ni el código de ¡Acepta el reto!

El sistema de logros consta de dos partes claramente diferenciadas, por un lado tenemos el ámbito del *framework*, donde está toda la lógica que nos proporciona la funcionalidad y el automatismo. Por otro lado tenemos una segunda parte, incluida en un paquete interno dentro del apartado de logros, en el cual tenemos las clases que implementan los logros individuales. Es este segundo grupo el que puede ser ampliado conforme se vayan integrando nuevos logros en el sistema.

Para la inclusión del sistema ha sido necesaria también la modificación de la base de datos, añadiendo una serie de tablas que nos permitan la persistencia, control de cambios, etc. de todo aquello relacionado con los logros.



Tablas para la persistencia de logros y sus relaciones.

Así, incluir un logro nuevo en el sistema supone añadir la tupla en la tabla de Achievements, rellenando todos los datos necesarios y especificando si es un logro manual o automático.

Exceptuando los logros manuales, que carecen de clase que especifique su comportamiento y que son asignados mediante la creación manual de la tupla en la tabla de AchievementsPerUser por parte de los administradores, también será necesaria la creación de la clase Java que define al logro, implementando en ella los métodos de actualizar y aplicar.

Para ilustrarnos tomemos de nuevo el caso del logro ***Siguiendo al conejo blanco***, el cual, para ser insertado en el sistema requirió incluir la tupla en base de datos y después crear una clase "DiezACs.java".

Dicha clase contendrá dos métodos, el de aplicarse desde cero, que hace un barrido en toda la base de datos y asigna el logro si procede y el que, suponiendo la entrada por argumentos de una lista de eventos, es capaz de extraer información de esos eventos y asignar el logro sólo a esos usuarios.

Añadir así los logros en el sistema, con ambos métodos, nos permite incluirlos con carácter retroactivo, pues el sistema detectará los logros recién insertados y los aplicará en toda la base instalada desde cero, dejando el contador del logro (dato que indica el último evento con el que fue comprobado) debidamente actualizado para, después, utilizar el segundo método en las siguientes comprobaciones.

Finalmente, el hecho de asignar un logro supone (bien por el sistema para los logros automáticos, bien manualmente por los administradores) la creación de una tupla en la base de datos, en la tabla de *Logros por usuario* (AchievementsPerUser) en la cual se especifica el binomio [Identificador del usuario | Identificador del logro], además de una marca de tiempo.

Antes de listar las clases del primer conjunto, aquel perteneciente al *framework*, tiene sentido explicar el funcionamiento y la dinámica del sistema de logros.

Cuando un usuario hace alguna interacción en el sistema, esto genera una serie de eventos, descritos en el capítulo anterior, y deja unas determinadas trazas, que pueden ser consultadas cuando se considere.

En nuestro servidor Apache + Tomcat, disponemos de un demonio que se activará cada cierto tiempo, este demonio realizará una llamada al gestor de logros (AchievementManager.java) y desencadenará todo el flujo de corrección, comprobación y asignación de logros, mediante una llamada al gestor de logros y a su método de auto actualizarse.

Dicho gestor lo que hará será un repaso al listado de logros disponibles, consultando, mediante la clase que sirve de enlace (AchievementDAO.java), la columna correspondiente al último evento registrado, comprobando y teniendo en cuenta así los últimos eventos. Si detectara que hay nuevos eventos desde la última vez que se comprobó un determinado logro, el gestor de logros será capaz de crear una instancia de ese logro, haciendo uso de la introspección y de la clase AchievementDAO.java para obtener los parámetros necesarios para dicha instanciación y actualizarlo, esto se hace logro por logro, para todos aquellos que consten en la base de datos y no sean manuales.

Un logro manual es aquel que consta en la base de datos y pude ser asignado, pero no tiene una clase java que represente su funcionamiento. La asignación es manual por parte de los administradores del sitio y está reservado para aquellos logros que se desee asignar por razones muy determinadas. Un ejemplo sería un usuario que informa de un error en la web y se desea premiar dicho comportamiento, que sería el logro: *Departamento de SQA*.

También puede ocurrir que encuentre un logro cuyo último evento, consultándolo desde el AchievementDAO.java, sea el evento cero. Esto implica que es un logro nuevo, que acaba de integrarse en el sistema y por ello, el gestor de logros, tendrá la capacidad de realizar una aplicación, y no una actualización, del logro sobre toda la base instalada.

La forma en que se reproduce todo este comportamiento será explicado explícitamente al describir cada una de las clases que intervienen en el proceso.

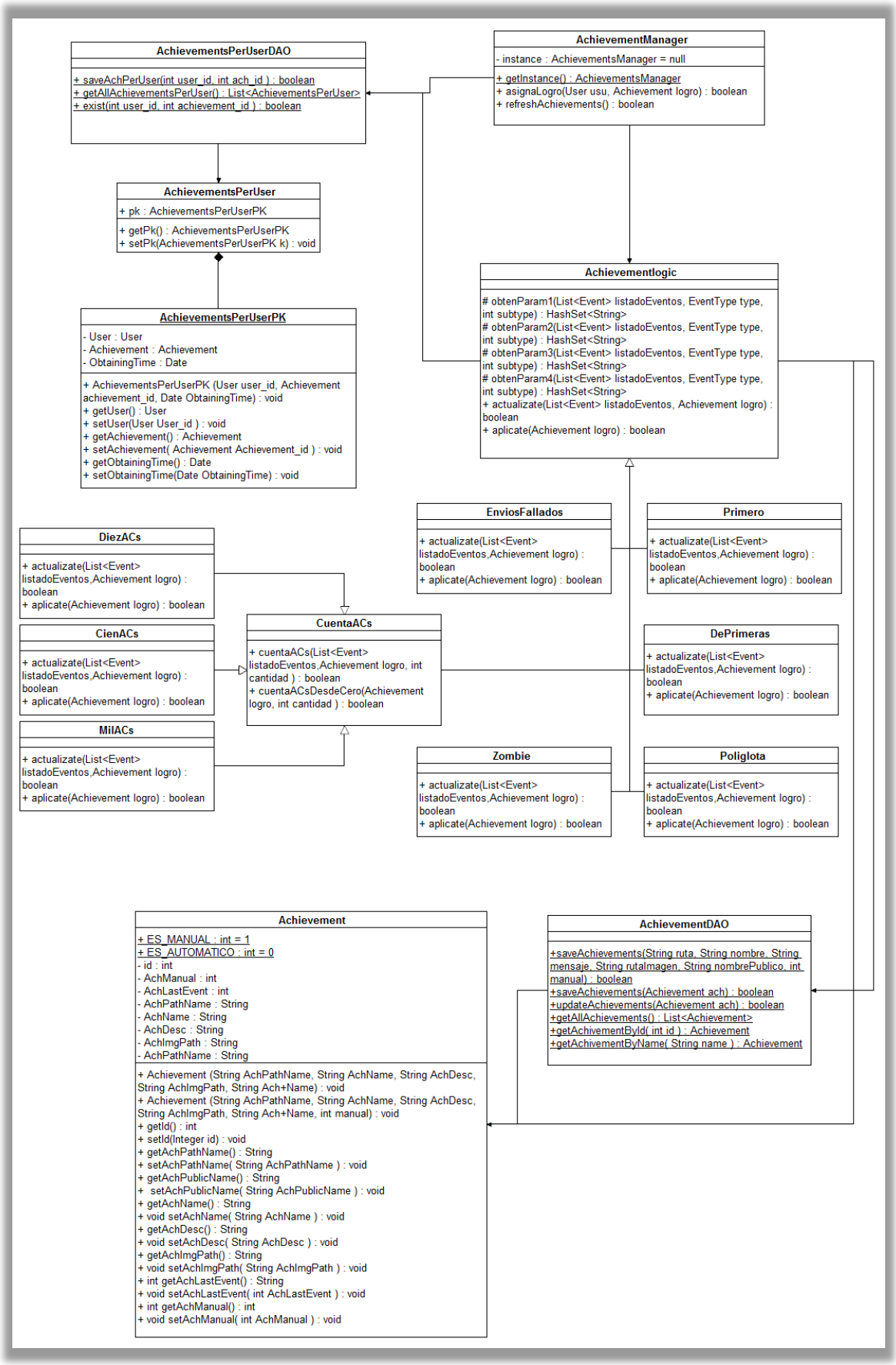


Diagrama de clases de los logros

Achievements.java

Es la clase que parametriza la entidad de los logros (de cara a Hibernate y la relación con la base de datos). Nos permite hacer instancias de los logros.

Cada logro consta de los siguientes atributos, debidamente parametrizados en la base de datos:

- **Id**, auto-incremental, será el identificador unívoco del logro.
- **AchPathName**, cadena de texto. Nos proporciona una ruta absoluta de la clase que representa este logro, permitiéndonos así, mediante introspección, crear una instancia en caso de necesitarla para poder acceder a los métodos propios de esta clase.
- **AchPublicName**, cadena de texto. Contendrá el nombre público del logro, aquel que será mostrado en la web, en las notificaciones, etc.
- **AchName**, cadena de texto. Representa el nombre interno del logro, es decir, el nombre de la clase que reproduce su comportamiento.
- **AchDesc**, cadena de texto. Nos proporciona la descripción del logro, aquella donde nos comunican la razón del logro. Será el texto que deberá mostrarse a los usuarios, junto con el nombre del logro, una vez consigan obtenerlo.
- **AchImgPath**, cadena de texto. Campo diseñado para almacenar la ruta del icono que representa a este logro.
- **AchLastEvent**, entero. Nos proporciona el identificador del último evento que fue comprobado para este logro. Mediante el uso de este campo podremos saber si necesitamos actualizar el logro (y desde que evento) o si necesitamos aplicarlo desde cero, al ser un logro nuevo.
- **AchManual**, entero. Nos proporcionará unos valores booleanos que indicarán si este logro es manual o no. En caso de ser manual, el gestor de logros lo ignorará durante sus comprobaciones.

AchievementDAO.java

Es la clase que nos permite el acceso a la tabla de logros. Coordinado con Hibernate, es el encargado de la transmisión de datos desde java a la base de datos en MySql, pudiendo así realizar las consultas necesarias, manipulación de los logros, obtener logros mediante un identificador, etc.

Al estar trabajando con esta metodología de modelo-vista-controlador podríamos fácilmente integrar otras formas de persistencia, pues todo queda debidamente encapsulado y sin acoplamientos.

Del mismo modo que ocurría con el EventsDAO.java, usar estas clases junto con el framework de Hibernate nos capacita para realizar, con un mínimo esfuerzo, todas aquellas consultas a la base de datos que podamos necesitar, directamente desde nuestras clases java.

AchievementsPerUser.java

Es la clase que parametriza la entidad de los logros por cada usuario (de cara a Hibernate y la relación con la base de datos). Esta clase nos permite almacenar los logros, representando así la asignación de un logro.

Contiene, dentro de ella, una subclase *embedded* (AchievementsPerUserPK) que es quien representa en memoria dinámica dicha asignación.

Así pues, esta clase, en memoria, sólo contiene un entero que representa la clave identificadora para su subclase, que es quien contiene los atributos y los métodos.

La combinación de ambas clases en memoria dinámica nos permite trabajar representar los datos almacenados en la base de datos, así como trabajar con el sistema de logros.

Contiene un único atributo:

- **Pk**, será el identificador unívoco del logro.

AchievementsPerUserPK.java

Esta subclase incrustada dentro de la clase AchievementsPerUser nos permite trabajar con los logros en memoria dinámica, utilizada en combinación con la clase anteriormente citada tenemos la capacidad para operar con las filas de la base de datos, obtener sus parámetros, etc.

La decisión de porqué implementar estas dos clases de forma separada en lugar de hacerlo en una única clase fue motivada por la necesidad de mantener una coherencia de datos y por eficiencia, así, primó la necesidad de guardar, como claves externas, los identificadores de usuarios y de logros.

Además, tener esta estructura nos permite diferenciar por un lado el almacenamiento en base de datos de los datos en memoria, que no contendrán identificadores de los usuarios y los logros, sino los propios logros y usuarios debidamente instanciados.

Cada elemento se compone de los siguientes atributos:

- **User**, instancia del usuario, pertenece a la clave primaria.
- **Achievement,** instancia del logro, pertenece a la clave primaria.
- **ObtainingTime,** Fecha y hora de asignación del logro.

AchievementsPerUserDAO.java

Es la clase que nos permite el acceso a la tabla de logros por usuario. Coordinado con Hibernate, nos permite trabajar, obtener y asignar los logros a los usuarios, una fila en esta tabla representa la asignación de un logro, pudiendo ser consultada en cualquier momento que sea necesario.

Como en los casos anteriores, trabajar con esta metodología de modelo-vista-controlador nos permite diferenciar el código, de la forma de mostrarlo y almacenarlo, pudiendo, en caso necesario, cambiar alguna de estas partes sin generar acoplamientos.

Esta clase, unida a Hibernate y a sus características propias de instanciación de los usuarios y los logros en lugar de obtener los identificadores nos capacita para trabajar, de forma muy eficiente y sin tener que salir del código de java con los datos almacenados en la base de datos.

AchievementLogic.java

Esta es una clase de la que heredarán los logros individuales. Tener esta estructura de herencia hace que nuestra funcionalidad sea viable, pues nos permite la instanciación por introspección, así como el almacenamiento, en memoria dinámica, mediante polimorfismo, de todos aquellos logros que sean necesarios en un determinado momento.

```
String nombreLogro = actual.getAchPathName();
Achievement logro = AchievementDAO.getAchivementByName(actual.getAchName());
try {
        Class<?> c = Class.forName(nombreLogro);
        Object o = c.newInstance();
        Achievementlogic logic = (Achievementlogic)o;
        System.out.println("Asignando: "+logro.getAchPublicName()+"...");
        if ( logic.aplicate(logro) ){
                actual.setAchLastEvent(contador);
                AchievementDAO.updateAchievements(actual);
} catch (Exception ex) {
        ex.printStackTrace();
}
```

Detalle de la instanciación y ejecución, mediante introspección, de los logros

Esta clase contiene algunos métodos genéricos, que serán usados por aquellos logros que lo necesiten, llamados *ObtenParamX*, donde la X representa el parámetro de los eventos (pasados en la llamada) que deseamos obtener. Estos métodos tienen sentido en la clase padre pues son comunes a todos los logros.

De este modo, para la aplicación de los logros, se instanciarán multitud de AchievementLogic, que serán en realidad los logros individuales.

También tenemos los métodos *Actualízate* y *Aplícate*, que serán implementados por las subclases, pues cada logro tendrá su propio código para aplicarse y actualizarse.

Los logros individuales que representan una especialización de la clase AchievementLogic y contienen, cada uno, una sobreescritura de los métodos *Actualízate* y *Aplícate*.

Dichas clases-logros son:

- DiezACs: Logro *Siguiendo al conejo blanco*.
- CienACs: Logro *Amo del calabozo*.
- MilACs: Logro *Gandalf de la programaci*
- Primero: Logro *Francotirador, un tiro, un muerto*.
- Zombie: Logro *Aparta zombie… que me distraes*.
- Políglota: Logro *Políglota*.
- EnvíosFallados: Logro *¡En plan vikingo!*
- DePrimeras: Logro *Aquí te pillo…Aquí te mato*.

Los logros manuales no están representados, pues como se mencionó, carecen de clase que los instancie y su existencia se limita a una tupla añadida en la tabla de logros.

Tenemos un caso particular de clase-logro que es el CuentaACs.

CuentaACs es una clase que hace de interface para tres logros individuales (DiezAC, CienACs y MilACs) establecida esta jerarquía podemos desarrollar una funcionalidad única para los tres logros, que son comunes y sólo varían en la cantidad de ejercicios resueltos que requieren para ser asignados.

AchievementManager.java

Esta clase es el core del sistema de logros. Es una clase Singleton que será llamada multitud de veces por el demonio lanzado desde Tomcat.

Es quien genera todo el flujo de ejecución del sistema de asignación de logros, pues, al recibir la invocación por parte del demonio de actualizar los logros, generará una serie de llamadas a los distintos métodos para que estos se actualicen, siguiendo sus propias normas de asignación.

Contiene un método llamado "asignaLogro" que sirve para crear una fila en la tabla de asignación de logros, habilitando así la opción de insertar manualmente logros en la base de datos. Esta opción sirve para aquellos logros, comentados anteriormente, que no son automáticos y que podrán asignarlos debido a méritos especiales de algún usuario, fuera del sistema, por ejemplo notificar un error en la web, etc.

Método refreshAchievements

Este es el método principal de la clase. La función es la de comprobar cuál es el número total de eventos. Seguidamente recorrerá, uno por uno, todos los logros existentes en la tabla de logros, comprobando, para cada uno de ellos, cuál ha sido el último evento actualizado.

Si el último evento registrado fue el cero se trata de un logro nuevo y por tanto lo que hará será crear una instancia del logro, usando introspección sobre el archivo AchievementLogic y cargando en él el logro que proceda, obteniendo el nombre y la ruta del logro desde la base de datos. Seguidamente llamará a ese objeto creado y le dirá que se actualice. Esto genera una aplicación del logro en el sistema.

Si el último evento registrado es distinto de cero, el logro ya existía en el sistema y por tanto debe ser actualizado desde el último evento que conste en el logro hasta el último evento registrado en el sistema.

Finalmente, actualizará los logros, individualmente, para poner su contador de evento actualizado al último recibido y a continuación hará un commit para persistirlo todo en el sistema.

8. Aplicación del sistema de logros

El objetivo del trabajo era la implementación del framework que permitiera la asignación de logros, pero dicho framework no está aún integrado porque requeriría trabajo en el frontend, lo cual se salía del alcance de este proyecto. Es por ello que ha sido necesario demostrar la validez del trabajo creado. Para ello se han generado dos clases que simulan dos comportamientos distintos y para dichas simulaciones se ha contado con la base de datos real de Acepta el reto, demostrando que el proyecto está listo para ser integrado, cuando se disponga de la funcionalidad en el frontend.

La primera de ellas genera un sistema idéntico al real de ¡Acepta el reto! en el cual, una vez cargados todos los envíos y establecido el sistema, aplicaremos nuestro sistema de logros y observaremos los resultados.

Con esta simulación se pretende obtener datos sobre qué ocurriría si se integrara el sistema con el estado actual de ¡Acepta el reto!

La segunda genera un sistema vacío y sobre él, empieza a realizar, en orden, los envíos, reproduciendo el uso real del sistema desde su origen. Durante la realización de estos envíos, y de forma aleatoria, se van haciendo llamadas al demonio que asigna los retos, comprobándolos y asignándolos.

Lo que se pretende obtener así es una ejecución simulada de cómo habría sido si el sistema de logros se hubiera aplicado desde el inicio de ¡Acepta el reto!

Es de importancia crítica mencionar que al finalizar ambas ejecuciones se constará de dos sistemas similares al real de ¡Acepta el reto! Con sus mismos envíos, usuarios, problemas, ejecuciones, correcciones, etc. es decir, una realidad paralela con el sistema de logros aplicado igualmente en ambos casos, con los mismos logros asignados a los mismos usuarios, demostrando así la corrección de ambos modos de ejecución, el de aplicación desde cero y el de actualización mediante el sistema de eventos.

Esto pone de manifiesto la realización satisfactoria del trabajo, pues se ha conseguido, con éxito, la inclusión del sistema funcional de logros, mostrándonos que en caso de incluirse el resultado de este trabajo en el sistema real, este sería el resultado y sería correcto.

8.1 Simulación

Para las simulaciones se hará uso de otra aplicación previa, llamada PueblaBD, que será la encargada de asentar ciertos pilares en nuestra base de datos MySql, estos son:

- Creación de 3500 usuarios, que simulan los 3500 usuarios reales de acepta el reto.
- Creación de 1200 problemas ficticios.
- Creación de los logros en la correspondiente tabla.

Dicha parametrización previa es necesaria dado que, por ejemplo, los logros serán añadidos manualmente por los administradores, cuando corresponda, de este modo esta clase reproduce ese comportamiento, al igual que la existencia de los problemas y los usuarios, que con datos inherentes a la web e independientes de nuestro trabajo, son simulados durante la ejecución de esta clase.

Tenemos entonces dos modos de ejecución para la simulación:

Puebla Envíos

Esta será la clase que, partiendo de la base establecida por el PueblaDB.java, nos realizará una inserción masiva de todos los envíos existentes en el historial de acepta el reto, planteando el sistema tal y como existe en la actualidad[3] .

A continuación, hará una llamada al demonio para que este llame al Gestor de logros y sean aplicados, desde cero, todos los logros.

Al finalizar dejará el sistema de logros integrado en el sistema, listo para continuar su ciclo de vida con los nuevos eventos que puedan surgir.

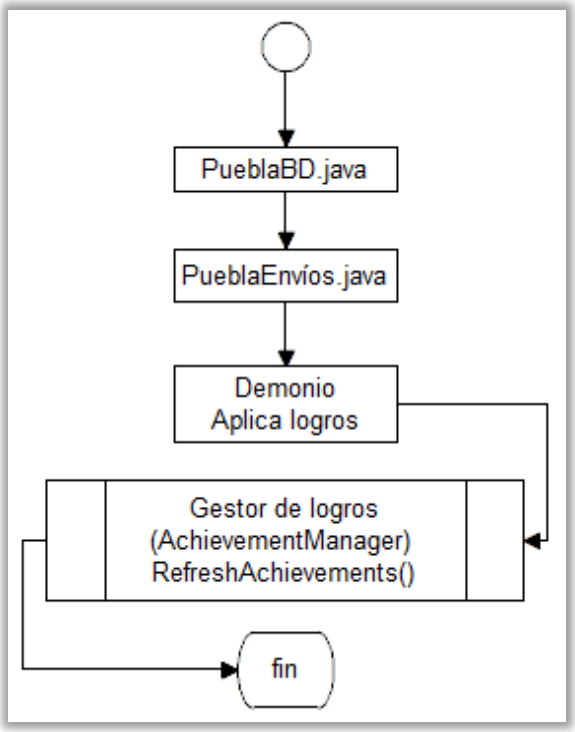


Diagrama de flujo de la ejecución

[3] Para ello se ha utilizado un fichero con la información de los 73035 envíos que tenía acepta el reto el día 15 de Junio de 2016

Simula Envíos

Esta será la clase que, partiendo de la base establecida por el PueblaDB.java, realizará, por orden cronológico, todos los envíos registrados en el sistema desde el origen del mismo.

Contiene, además, un contador aleatorio, mediante el cual, tras cada envío, existe cierta posibilidad de llamar al demonio[4], para que este actualice el sistema de logros.

En la primera llamada al demonio el tiempo de ejecución será mayor, pues todos los logros constarán como "de nueva inserción" y habrá que aplicarlos, tras esta primera llamada, todas las siguientes consistirán en una actualización, requiriendo, por tanto, un tiempo ínfimo para comprobarse. Finalmente se realizará una última llamada al demonio para evitar que queden eventos pendientes.

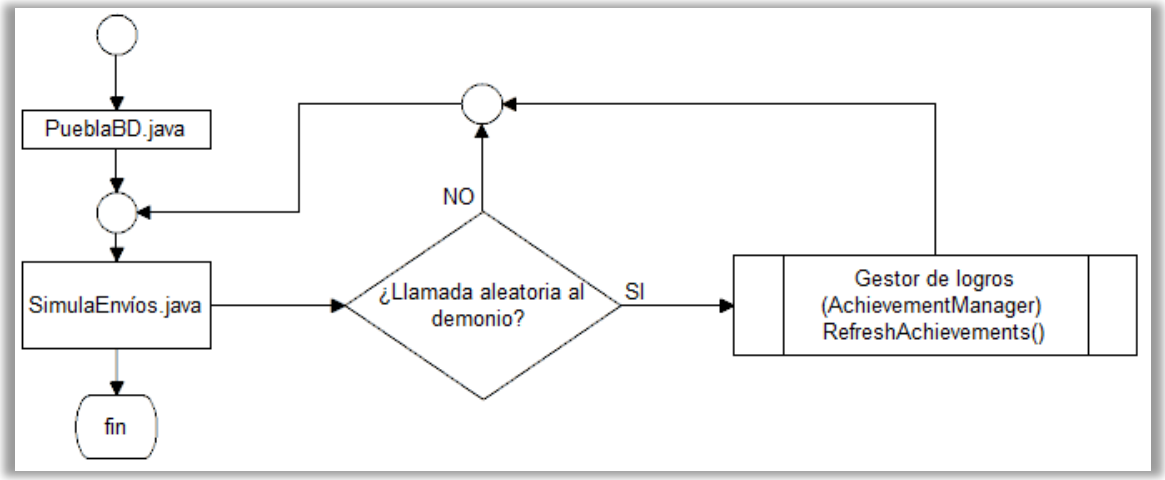


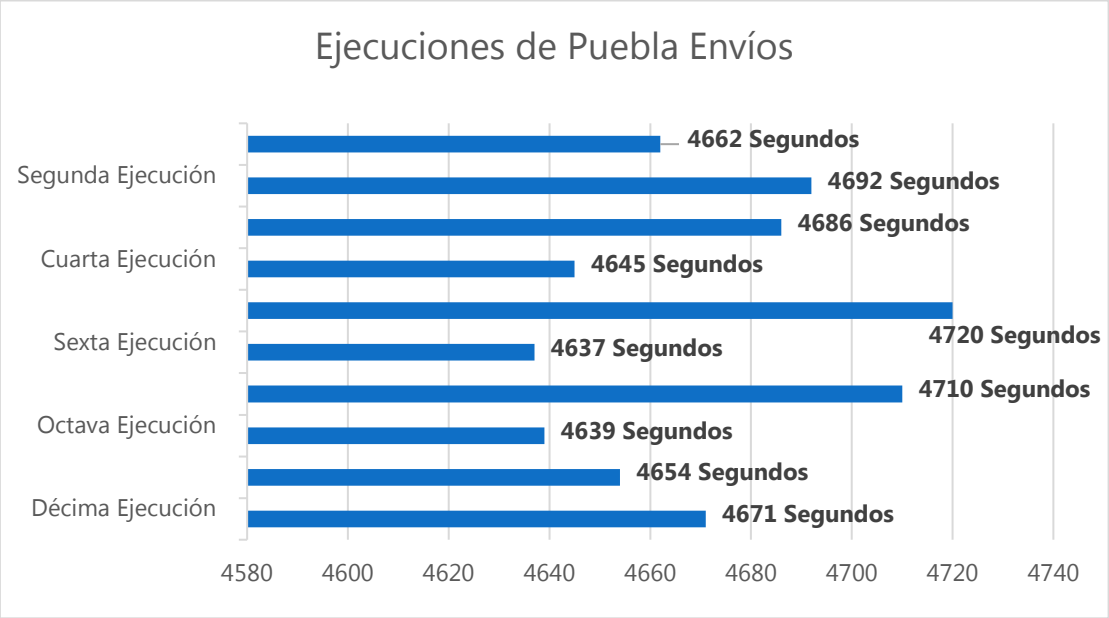
Diagrama de flujo de la ejecución

Al finalizar, exactamente igual que ocurría en el caso anterior, y como demostración de que se han cumplido los objetivos, dejará el sistema de logros integrado en el sistema, listo para continuar su ciclo de vida con los nuevos eventos que puedan surgir.

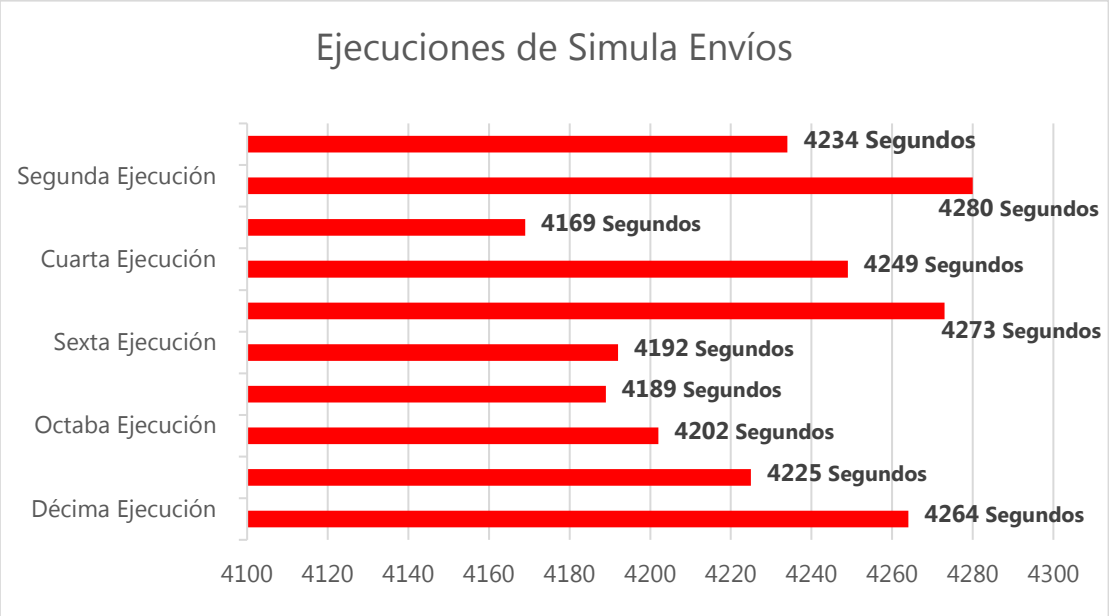
[4] Se comprueba empíricamente que dicha llamada ocurre en un rango que va desde los cero a los treinta envíos.

8.2 Validación

Se han realizado 10 ejecuciones completas para cada una de los modos de simulación. Los tiempos medidos se muestran a continuación.



Valor medio obtenido: 4671,6 segundos.



Valor medio obtenido: 4227,7 segundos.

Realizadas las simulaciones sobre el sistema, hemos llegado al punto en el cual debemos validar nuestro trabajo. Para ello analizaremos los logros asignados[5], tiempo consumido, etc. demostrando así, el resultado satisfactorio.

Analizando los tiempos

La medida de tiempos se ha hecho sólo de la parte de aplicación del sistema de logros, así, para el "PueblaEnvios", sólo se ha medido el tiempo de actuación del demonio.

Lo mismo sucede con "SimulaEnvíos", donde se ha tomado nada más el tiempo de ejecución del demonio, acumulando cada pequeña actualización de los tiempos e imprimiéndola al finalizar.

De las anteriores graficas de tiempos podemos extraer la siguiente información: aunque el tiempo varía de una ejecución a otra, cosa que es inherente a la informática y a los sistemas operativos[6], se puede observar que en ambos casos está bastante acotado. También podemos decir, de forma decisiva, que el sistema de actualización mediante eventos es más eficiente, en tiempo, que el sistema de aplicación en toda la base instalada. Además pone de manifiesto otra realidad: mientras el tiempo consumido en la ejecución del primer sistema, en el cual aplicamos para toda la base, aumenta de forma exponencial con cada logro insertado en el sistema, el segundo modo de funcionamiento, con actualizaciones y mediante el sistema de eventos, apenas nota diferencia al incluir nuevos logros, pues las comprobaciones son sólo sobre los eventos recibidos, que puedan afectar al logro, y estaríamos manejando unas cifras de milisegundos.

Un análisis más profundo revela que la aplicación en base instalada conlleva un tiempo **T1** (Se ha comprobado que el tiempo medio de aplicación del sistema de eventos en una base instalada es de aproximadamente 4200-4700 segundos) y la actualización mediante el sistema de eventos supone un tiempo **T2**[8] (Se observa que cada actualización de los logros ya aplicados en la base de datos conlleva unos 4-5 segundos)**.** Es, por tanto, debido a los tiempos anteriormente mencionados, que podemos demostrar que **T1** es aproximadamente del orden de **mil** veces mayor que **T2.**

[5] Las asignaciones individuales serán tratadas en el apartado siguiente, Logros Implementados.
[6] Procesos en segundo plano, actualizaciones del sistema, comprobaciones de Windows, son sucesos que afectan a la ejecución de nuestra aplicación.
[8] Dicho Tiempo **T2** hace referencia al tiempo que se tarda en "actualizar", para unos logros ya "aplicados" con anterioridad.

Supongamos la inexistencia del sistema de eventos, pero si del sistema de logros, que es aplicado a primera hora de la mañana por primera vez, pero pasará a estar integrado y por tanto a ser comprobado cada cierto tiempo. Tendríamos únicamente la opción de aplicar los logros para toda la base en cada actualización, supongamos 48 actualizaciones diarias: una cada 30 minutos [9], así:

Coste diario en tiempo de dicha ejecución:

Primer día: T1 * 48 actualizaciones **= Inmanejable**

Siguientes días: T1 * 48 actualizaciones **= Inmanejable**

Dado que cada actualización supondría reaplicar de nuevo en toda la base instalada, dicho sistema requeriría, para su funcionamiento, más horas de las disponibles al día, además de ser igual todos los días. Tiempo que además aumentaría conforme más logros, usuarios y envíos fueran incluidos en el sistema y que harían inviable el sistema de logros.

Supongamos ahora el segundo caso, en el cual si disponemos del sistema de eventos, y por tanto de la función de actualizar con el tiempo anteriormente mencionado **T2**.

Para el mismo caso anterior, el coste diario de la ejecución seria:

Primer día: T1 + (T2*48 actualizaciones**)**

Siguientes días: (T2*48 actualizaciones**)**

El gasto en tiempo, total, diario de aplicar el sistema de logros sería de unos 240 segundos, es decir, aproximadamente un 0,2% del día. Evidentemente este tiempo también escalaría conforme se vayan incluyendo logros en el sistema, pero una vez aplicado (una única vez) el logro en el sistema, el tiempo de actualización prácticamente no variaría, apenas milésimas por cada logro nuevo.

Es por todas estas razones que se considera un éxito la implantación del sistema de eventos como complemento del sistema de logros y una condición *sine qua non* para que su existencia sea factible.

[9] Cantidad muy inferior a los valores previsiblemente reales, que serían aproximadamente 128 comprobaciones diarias, dado que para los usuarios 30 minutos es demasiado.

Analizando los datos

Respecto a las salidas obtenidas, podemos confirmar que el proyecto funciona correctamente, pues independientemente de si hemos ejecutado un modo u otro, el total de asignaciones para cada uno de los logros es igual.

Tampoco varían dichas asignaciones al ejecutar la aplicación de nuevo. Así, observando el total de ejecuciones medidas (diez para cada simulación) comprobamos que varían los tiempos, pero los logros asignados son, las veinte veces, los mismos y a los mismos usuarios.

Dicho de otra forma: si aplicando en base instalada hay un usuario que obtiene un determinado logro, simulando el sistema mediante envíos y el sistema de eventos comprobaremos que ese mismo usuario obtiene ese mismo logro.

Por tanto, la asignación de los logros es totalmente correcta, satisfaciendo así los requisitos del proyecto, pues era condición necesaria que la implementación del sistema de logros fuera, una vez aplicado, equivalente a llevar aplicado desde el inicio de la web.

8.3 Logros implementados

Siguiendo al conejo blanco

El logro *Siguiendo al conejo blanco* es un logro que se asignará a aquellos usuarios que hayan realizado satisfactoriamente diez ejercicios distintos.

Amo del calabozo

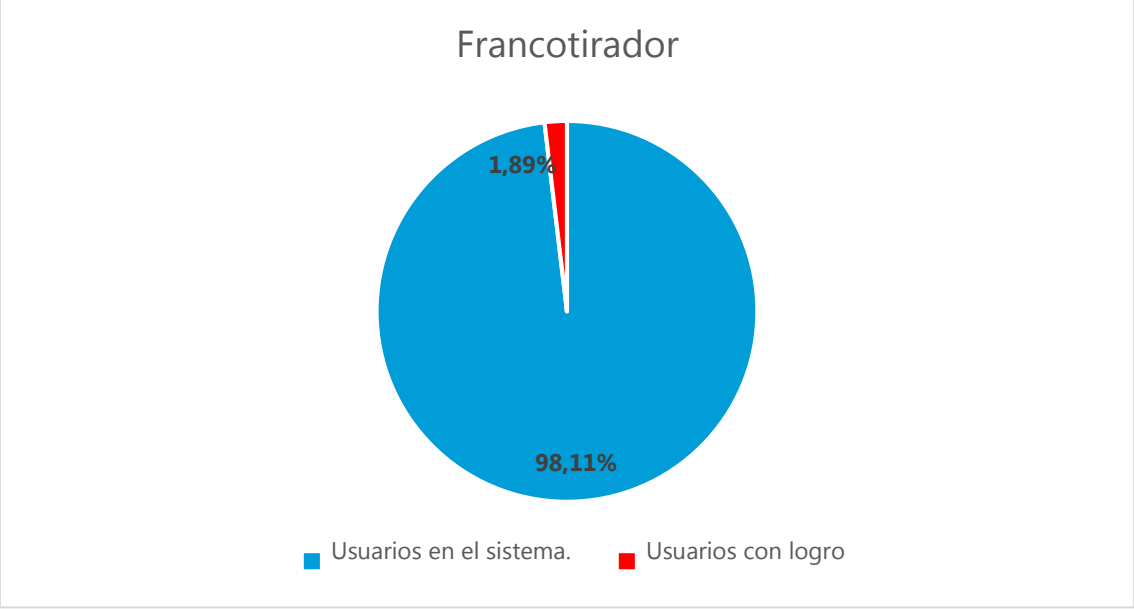
El logro *Amo del calabozo* es un logro que se asignará a aquellos usuarios que hayan realizado satisfactoriamente cien ejercicios distintos.

Gandalf de la programación

El logro *Gandalf de la programación* es un logro que se asignará a aquellos usuarios que hayan realizado satisfactoriamente mil ejercicios distintos. (¡aún no se dispone de tantos problemas!)

Francotirador

El logro *Francotirador* es un logro que se asignará a aquellos usuarios que hayan sido los primeros en resolver un problema.

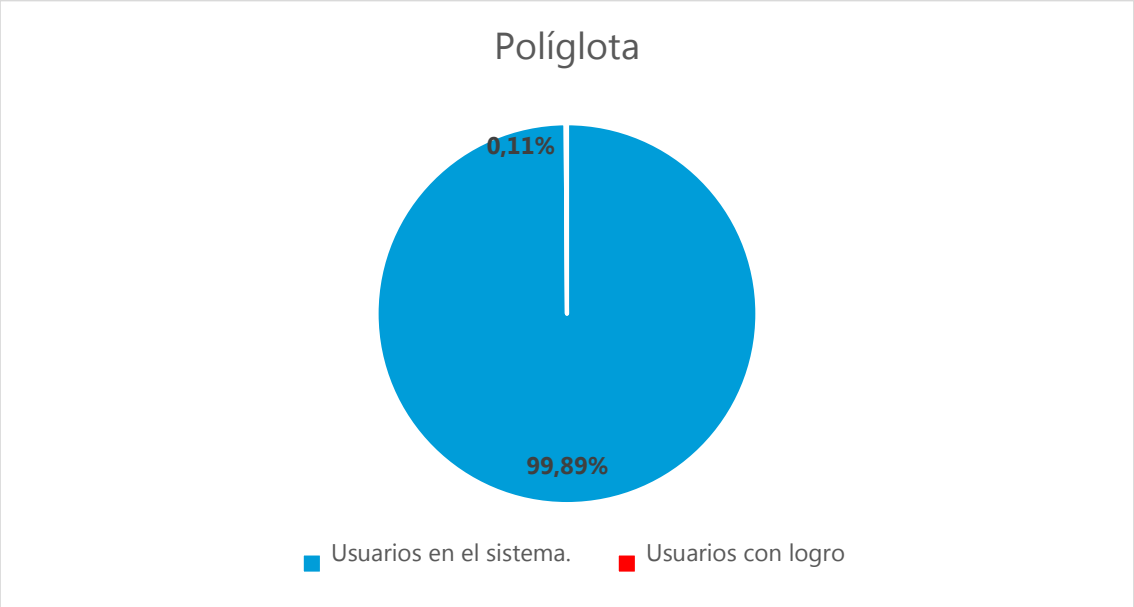


¡En plan Vikingo!

El logro *¡En plan vikingo!* lo obtendrán aquellos usuarios que hayan enviado diez ejercicios seguidos incorrectos (Bien por límite de tiempo, errores en el código, etc.)

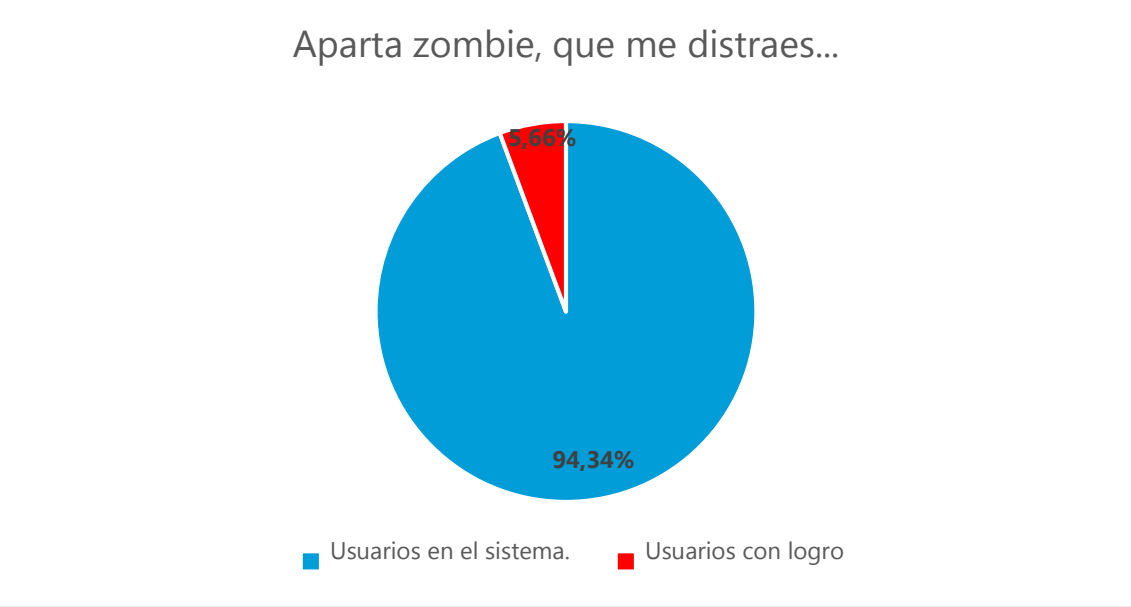
Políglota

El logro *Políglota* es un logro que será obtenido por aquellos usuarios que hayan resuelto satisfactoriamente diez problemas distintos en cada uno de los lenguajes disponibles: C, C++ y Java.



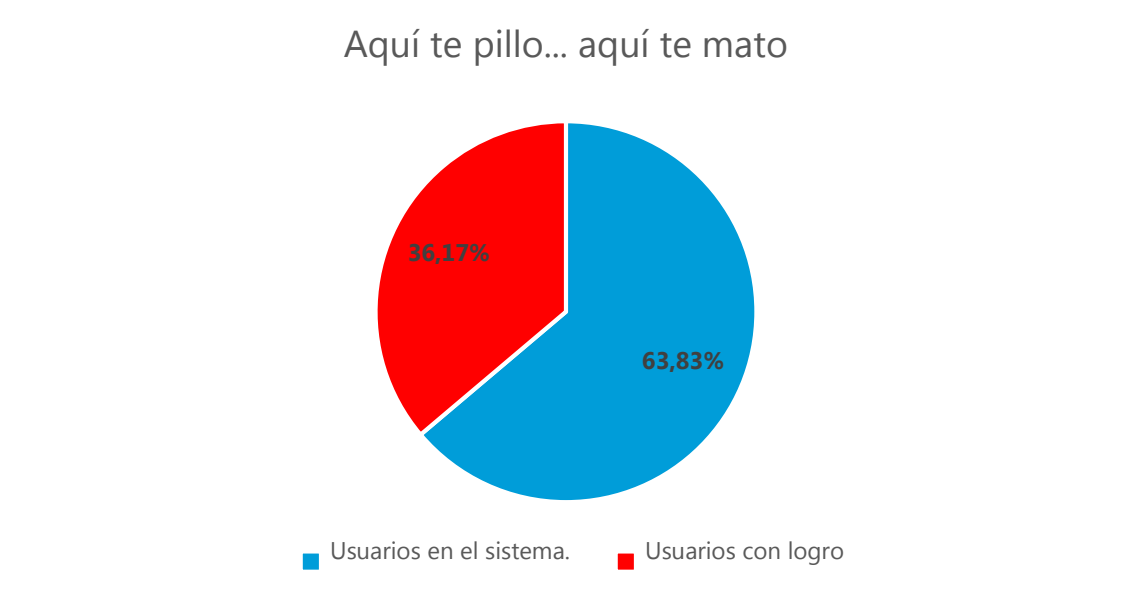
Aparta zombie, que me distraes…

El logro *Aparta zombie, que me distraes…* es un logro que será obtenido por aquellos usuarios que realicen un envío correcto a determinadas horas de la madrugada.



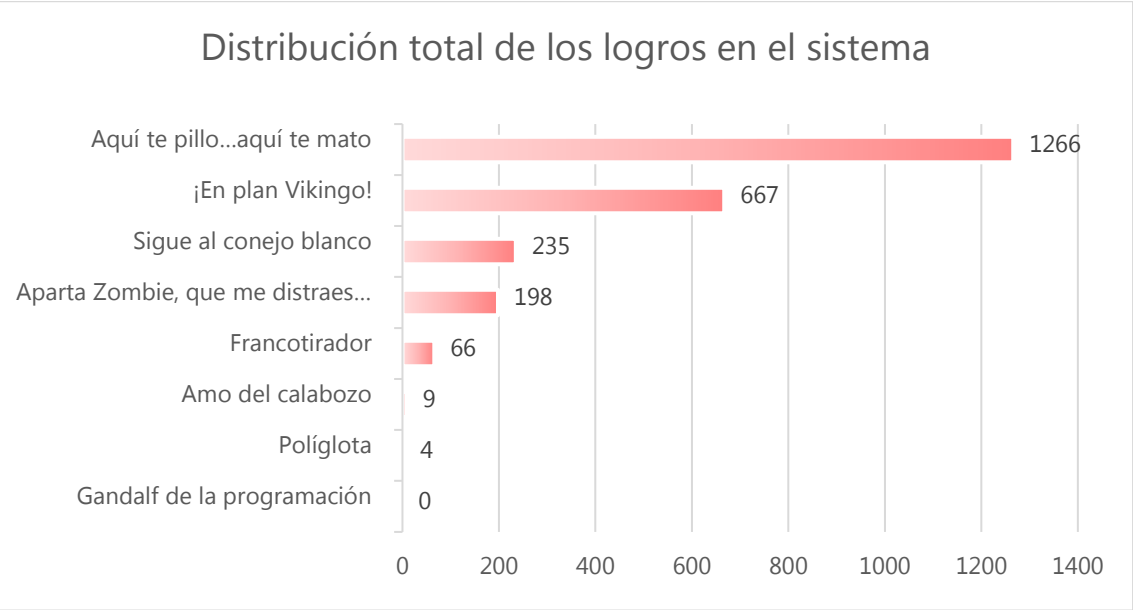
Aquí te pillo…aquí te mato.

El logro *Aquí te pillo…aquí te mato* es un logro que se obtiene al resolver un ejercicio (obtener un AC) en el primer intento.



Resumen general de la asignación de logros

Dicha distribución se corresponde tanto con la ejecución en base instalada como en la ejecución mediante el sistema de eventos y actualizaciones, los resultados son los mismos.



Como observación personal, se observa como gran parte de los usuarios logran solucionar algún problema con el primer envío (un 36%) lo cual tiene sentido, pues al proporcionarnos la página un enunciado con entradas y salidas, podemos reproducir el ejercicio en nuestra máquina y sólo cuando sabemos que funciona, enviarlo.

9. Conclusiones

Con el proyecto terminado se puede afirmar que se han cumplido los objetivos, pues nos encontramos con un desarrollo firme, que cumple con todos los parámetros necesarios en una buena aplicación, respetando todos y cada uno de los requisitos funcionales relatados en la introducción de la memoria.

Se ha obtenido un sistema eficiente en tiempo y memoria, capaz de incluirse dentro del *backend* del servidor, que trabaja de forma autónoma y paralela al resto de procesos, transparente para los usuarios y los administradores. Un sistema capaz de, automáticamente, comprobar la necesidad y los requisitos de asignación de logros a los usuarios que proceda.

Así mismo se ha desarrollado un sistema capaz de ser ampliado con un mínimo de esfuerzo por parte de los desarrolladores y lo suficientemente cohesionado como para no impactar en el resto de la funcionalidad de la web si fuera retirado de forma abrupta.

Mediante la aplicación del sistema y la siguiente validación, consecuencia de las simulaciones realizadas sobre datos reales de la página web, podemos afirmar que el proyecto se encuentra completamente operativo y listo para ser integrado en la página web, de modo que, desde el momento de la aplicación, realice su función de inclusión y asignación de logros de manera efectiva.

9.1 Valoración personal

Estoy muy satisfecho de la realización de este proyecto, pues me ha permitido poner a prueba algunos de los conocimientos aprendidos durante la carrera.

Encontrarme con dificultades como el tiempo desproporcionado que tardaba en aplicarse en toda la base instalada el sistema de logros, para después encontrar una solución óptima, como fue la inclusión del sistema de eventos, que nos permite aplicar todo el sistema de logros en apenas dos segundos, con una total efectividad ha sido bastante gratificante.

También ha sido muy interesante el desarrollo de la implementación individual de los logros, pues ha requerido de una aplicación directa de directrices y formas de implementar aprendidas en las asignaturas de *Estructura de algoritmos* así como de *Técnicas algorítmicas de la ingeniería del software*.

Muy importante ha sido la diferenciación entre la aplicación de un logro y la actualización del mismo, pues al actuar con parámetros distintos (uno con toda la base de datos y la otra con los eventos) suponía un reto realizarlo de forma que otorgara los mismos logros, sin importar el origen de los datos. Esto ha dado bastantes problemas de cara a la implementación pero estoy muy satisfecho con el trabajo realizado, pues finalmente se ha conseguido el objetivo de que la asignación de logros sea similar (condición que era requisito funcional del proyecto).

Finalmente puedo afirmar que se han aplicado los principios básicos aprendidos en asignaturas como *Ingeniería del software y Modelado de software* al obtener una aplicación que es, no solo altamente cohesiva y bajamente acoplada, sino totalmente escalable.

9.2 Trabajo futuro

Durante la realización del proyecto se han puesto de manifiesto una serie de opciones que puede ser interesante integrar en el futuro, a saber:

- Integración del sistema de logros dentro del *frontend* de la web, de forma que sea accesible desde el perfil de los usuarios.
- Posibilidad de conectar, mediante servicios web, nuestro perfil en ¡Acepta el reto! con otros perfiles existentes en otros jueces online, obteniendo estadísticas sobre el uso y la realización de ejercicios en todos ellos.
- Integración de ¡Acepta el reto! en las diferentes redes sociales, permitiéndonos publicar, mediante servicios web, nuestros ejercicios, resultados, logros, etc. en redes sociales como puedan ser Facebook o Twitter.
- Creación de una plataforma móvil que permita acceder a la web y a su funcionalidad desde dispositivos móviles.
- Creación de un sistema de envío de ejercicios entre usuarios.
- Ampliación del sistema de rankings, para una consulta con más detalle y eficiente.
- Integración en la web de una serie de estadísticas y gráficas, obtenidas de la base de datos, que se actualice de manera autónoma y nos permita obtener datos sobre nuestro ciclo de vida, como usuarios, dentro de la web.

10. Conclusions

The finished project can claim to have met the objectives, because we have a strong development, which meets all necessary parameters in a sound application, respecting each and every one of the functional requirements described in the introduction of the memory .

We have obtained an efficient system in terms of time and memory, capable of being included within the backend server, which works autonomously and in parallel to other processes, transparent to users and administrators. A system that can automatically verify the necessity and requirements to assign achievements to appropriate users.
We have also developed a system that can be expanded with minimal effort by developers and cohesive enough to not impact the rest of the functionality of the web if it were withdrawn abruptly.

By applying the system and its following validation, as a result of the simulations on actual data of the website, we can declare that the project is fully operational and ready to be integrated in the website, in such a way that from the very moment of its application, it can perform its activity of inclusion and assignment of achievements effectively.

10.1 Personal assessment

I am very satisfied with this project, as it has allowed me to test some of the knowledge acquired during my studies.

It has been deeply gratifying to discover hurdles such as the disproportionate amount of time it took to apply the entire base of the achievement system, and to then find an optimal solution, as was the inclusion of system events, which allows us to apply the whole achievement system in just two seconds with complete effectiveness.

It has also been very interesting to develop the individual implementation of the achievements, since it has required direct application of guidelines and ways to implement learned in the subjects of Algorithm structure and algorithmic techniques of software engineering.

It has been very important to differentiate between the application of an achievement and updating it, because acting with different parameters (one with the entire database and the other with events) meant a challenge so that it granted the same achievements , regardless of the source of the data. This has created many problems in terms of the implementation but I am very satisfied with the work done, as at the end of the day, it has achieved the goal of the assignment allocation being similar (a condition that was functional requirement of the project).

Finally I can say that basic principles learned in subjects such as software engineering and software modeling have been applied in order to obtain an application that is not only highly cohesive and not too coupled, but fully scalable.

10.2 Future work

Project development has revealed a number of interesting options that can be integrated in the future, namely:

- Integration of the achievements system within the web frontend, so that it is accessible from a user profile.
- The ability to connect through web services, our profile in Accept the challenge! with other existing profiles of online judges, obtaining usage statistics and conducting exercises in all of them.
- Integration of Accept the challenge! in different social networks, allowing us to publish, using web services, our exercises, results, achievements, etc. on social networks like Facebook or Twitter.
- Creation of a mobile platform to access the website and its functionality from mobile devices.
- Creating a delivery system for exercises between users.
- Expansion of the rankings system for a more detailed and efficient query.
- Integration into the web of a series of statistics and graphs, obtained from the database, which are updated autonomously and allow us to obtain data on our life cycle, as users within the web.

11. Bibliografía

[1] «Wikipedia: Sistema de logros,» [En línea]. Available: https://es.wikipedia.org/wiki/Logro_(videojuegos).

[2] "Wikipedia: Nonlinear Gameplay," [Online]. Available: https://en.wikipedia.org/wiki/Nonlinear_gameplay.

[3] "Ubisoft: Assassins Creed," [Online]. Available: https://www.ubisoft.com/en-US/game/assassins-creed/.

[4] «Ubisoft: Far Cry 3,» [En línea]. Available: https://www.ubisoft.com/es-ES/game/far-cry-3/.

[5] "Grand Theft Auto," [Online]. Available: http://www.rockstargames.com/grandtheftauto/.

[6] «The Elder Scrolls,» [En línea]. Available: http://www.elderscrolls.com/.

[7] «Steam,» [En línea]. Available: http://store.steampowered.com/.

[8] «Origin,» [En línea]. Available: https://www.origin.com/es-es/store/.

[9] «PlayStation Network,» [En línea]. Available: https://www.playstation.com/es-es/explore/playstation-network/.

[10] «Google Play,» [En línea]. Available: https://play.google.com/store.

[11] "Stackoverflow," [Online]. Available: http://stackoverflow.com/.

[12] «C++,» [En línea]. Available: http://www.cplusplus.com/.

[13] «Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Java_%28lenguaje_de_programaci%C3%B3n%29.

[14] «TopCoder,» [En línea]. Available: https://www.topcoder.com/.

[15] «Wikipedia: Google,» [En línea]. Available: https://es.wikipedia.org/wiki/Google.

[16] «URI online judge,» [En línea]. Available: https://www.urionlinejudge.com.br.

[17] «UVa Online Judge,» [En línea]. Available: https://uva.onlinejudge.org/.

[18] "Sphere Online Judge," [Online]. Available: http://www.spoj.com/.

[19] «Acepta el Reto,» [En línea]. Available: https://www.aceptaelreto.com/.

[20] "Wikipedia: HTML," [Online]. Available: https://en.wikipedia.org/wiki/HTML.

[21] "Javascript," [Online]. Available: https://www.javascript.com/.

[22] "PHP," [Online]. Available: http://www.php.net/.

[23] L. P. Richard Wiener, Fundamentals of OOP and data structures in Java, Cambridge: Cambridge University Press, 2000.

[24] R. F. a. T. O. James Elliott, Harnessing Hibernate, Sebastapol: O'Reilly, 2008.

[25] P. DuBois, MySQL, Madrid: Anaya, 2005.

[26] «Apache Software Foundation,» [En línea]. Available: https://www.apache.org/.

[27] «Tomcat,» [En línea]. Available: https://tomcat.apache.org/.