Universidad Central de Venezuela

Facultad de Ciencias

Escuela de Computacion

CONSTRUCCIÓN DE UN JUEZ PARA COMPETENCIAS DE PROGRAMACIÓN

Br. Emilio Tirado

Br. Ricardo Tovar

Prof. Hector Navarro, Tutor

Caracas, 18 de Mayo del 2015

Universidad Central de Venezuela

Facultad de Ciencias

Escuela de Computacion

CONSTRUCCIÓN DE UN JUEZ PARA COMPETENCIAS DE PROGRAMACIÓN

Br. Emilio Tirado

Br. Ricardo Tovar

Prof. Hector Navarro, Tutor

Caracas, 18 de Mayo del 2015

CONSTRUCCIÓN DE UN JUEZ PARA COMPETENCIAS DE PROGRAMACIÓN

Br. Emilio Tirado
Br. Ricardo Tovar

Trabajo Especial de Grado presentado

ante la ilustre Universidad Central de Venezuela

como requisito parcial para optar al título de

Licenciado en Computacion.

Prof. Hector Navarro, Tutor

18 de Mayo del 2015

Quienes suscriben, miembros del Jurado designado por el Consejo de la Escuela de Computacin que examinó el trabajo presentado por el **Br. Emilio Tirado, C.I. 19514240** y el **Br. Ricardo Tovar, C.I. 19967755**, titulado: "**Construcción de un juez para competencias de programación**" para optar al título de Licenciado en Computacion, dejan constancia de lo siguiente:

Leído como fue, dicho trabajo por cada uno de los miembros del Jurado, se fijó el día 18 de Mayo de 2015 a las 14:00 horas, para que sus autores lo defendieran en forma pública, lo que se hizo en el Centro de Computación Gráfica de la Escuela de Computación, en la Facultad de Ciencias de la Universidad Central de Venezuela, mediante una exposición oral de su contenido, luego de lo cual respondieron las preguntas formuladas por el Jurado y público en general. Finalizada la defensa pública del Trabajo Especial de Grado, el Jurado decidió aprobarlos.

En fe de lo cual se levanta la presente acta, en Caracas a los dieciocho días del mes de Mayo de dos mil quince, dejándose también constancia de que actuó como Coordinador del Jurado, el Profesor Tutor Héctor Navarro.

Prof. Hector Navarro, Tutor — 18 de Mayo del 2015

Prof. Eugenio Scalise — 18 de Mayo del 2015

Prof. Esmitt Ramirez — 18 de Mayo del 2015

Caracas, 18 de Mayo del 2015

Dedicamos este trabajo a nuestras madres Marta y Mónica que soñaban con vernos graduados y se esforzaron para que lograramos esta meta.

Lo dedicamos también a Trino Gómez, cuya amistad desde que ingresamos a las competencias de programación ha sido muy importante para nosotros, sin él no hubiesemos tenido la oportunidad de disfrutar todas esas competencias.

Para finalizar dedicamos este trabajo al mundo de las competencias de programación, gracias a éste hemos crecido personal y profesionalmente a lo largo de nuestra carrera.

Agradecimientos

Agradecemos a las personas que nos ayudaron y apoyaron durante la realización de este trabajo especial de grado, la profesora Marta Montero por proveernos de un lugar donde realizar dicho trabajo, al compañero Juan Nieto por el asesoramiento gráfico realizado, así como por la elaboración del logo de la pieza de software.

También a las personas que colaboraron en la etapa de levantamiento de requerimientos: Trino Gómez, Hector Navarro, Walter Hernandez, Esmitt Ramírez y todos los competidores que participaron en nuestro maratón de prueba.

En especial agradecemos a nuestras madres Marta y Mónica y demás familiares por todo el apoyo dado durante nuestras vidas y en particular durante esta carrera. Sin ellas nada hubiera sido posible.

Para culminar agradecemos a Dios por la fortaleza y paciencia que nos otorgó para realizar este trabajo.

Índice General

Introducción

Desde hace algunos años con el desarrollo de las Ciencias de la Computación, han ido en aumento de la misma manera el número de competencias vinculadas a esta rama de las ciencias.

Comenzando con las competencias de ACM International College Programming Contest hacia finales de los años 70, luego con la Olimpíada Internacional de Informática hacia finales de los 80, el número de competencias ha ido en aumento, gracias al desarrollo de Internet, hoy en día existen varios sitios que realizan competencias regularmente como TopCoder y Codeforces, competencias mundialmente reconocidas como Google CodeJam y Facebook Hacker Cup, cada una de ellas con sus reglas particulares para darle un toque personal al evento que quieren realizar. En el primer capítulo hacemos una compilación de los aspectos importantes de las reglas de cada competencia.

El desarrollo de las Ciencias de la Computación, no solo vincula a las competencias de programación, cada día nacen nuevas tecnologías, frameworks, nuevas técnicas para hacer las cosas, en fin, mejoras en el software que buscan hacer el desarrollo de sistemas más rápido, sencillo y elegante para el programador. En nuestra experiencia, conducir una competencia de programación involucra muchos aspectos, y uno de ellos es contar con un software robusto, eficiente y fácil de configurar, que permita adaptarse a los diversos formatos de competencia, y permitir a los jueces y competidores enfocarse en la misma. En el capítulo 2 se presentan los jueces existentes en la actualidad, así como experiencias personales con el uso de los mismos, y adicionalmente en el capítulo 3 se

presentan algunas de las tecnologías presentes en la actualidad, que hemos utilizado en la elaboración de un sistema de estas características.

Por último en el capítulo 4, realizamos una propuesta para la construcción de un sistema, que pueda permitir a los organizadores de un evento de programación, llevar a cabo su evento con las reglas que consideren convenientes para su realización, todo ello cumpliendo con las características de ser robusto, eficiente y permitiendo a los usuarios, tanto jurados como competidores, una experiencia sencilla a la hora de realizar su competencia de programación.

Capítulo 1

Competencias de Programación

Las competencias de programación, son eventos en los cuales, los participantes son probados en la resolución de problemas de naturaleza algorítmica. Los problemas son presentados mediante un enunciado, con una entrada bien definida y se especifica la salida que debe producir el competidor con su algoritmo de solución.

Tienen su origen hacia los años 70 con la aparición de las competencias de la ACM-ICPC, unos años más tarde se sumaría la Olimpíada de Informática como las competencias de programación presenciales de mayor importancia. Adicionalmente a éstos dos eventos, con el auge de Internet, han aparecido competencias a nivel mundial como TopCoder, Google CodeJam, Facebook Hacker Cup, y Codeforces, que han propiciado un desarrollo a nivel mundial en cuanto a la cantidad y calidad de las competencias.

En las siguientes secciones se describen en detalle las competencias más importantes, tanto las presenciales como las que se realizan en la actualidad a través de Internet.

1.1. ACM International Collegiate Programming Contest

El ACM International Collegiate Programming Contest (ICPC), provee a los estudiantes universitarios oportunidades para interactuar con estudiantes de otras uni-

versidades, para desarrollar y demostrar sus habilidades para trabajar en equipo, y solucionar problemas, así como sus habilidades de programación. La competencia le provee una plataforma a la industria de computación ACM, y a la academia, para impulsar y atraer las miradas del público a las nuevas generaciones de profesionales en las ciencias de la computación que buscan la excelencia, como se expresa en la misión del evento [1].

En las siguientes secciones se describe en detalle toda la información necesaria del evento, su formato de competencia, condiciones para participar, los problemas, evaluación y puntaje como se expresa en las reglas tanto para los regionales [4] como la final mundial[1] y las políticas del evento [2].

1.1.1. Formato de la Competencia

El ICPC es una competencia realizada en dos rondas, entre equipos de estudiantes representando sus casas de estudio de educación superior. Los equipos deben competir primero en Competencias Regionales, de las cuales, los equipos con los puntajes más altos avanzan a la Final Mundial ACM-ICPC. Como se determinó por el comité ejecutivo del ICPC, las Competencias Regionales están agrupadas en Super Regiones. Cada Super Región está conformada por un conjunto de regiones, y éstas a su vez por los países. El día de la competencia, participan los equipos de una Super Región completa, sin embargo, los cupos son asignados según las regiones que conforman la Super Región.

Los campeones de los Super Regionales, y los campeones mundiales reciben un reconocimiento en la Final Mundial, donde se entregan medallas de oro, plata y bronce.

1.1.2. Participantes

Cada equipo estará conformado por 3 competidores, los cuales, deben ser elegibles para participar en la Final Mundial ICPC, como se describe más adelante.

Un representante de cada institución, típicamente un miembro de la facultad, debe servir como el entrenador designado por el equipo. El entrenador debe certificar la elegibilidad de los miembros del equipo, y sirve como contacto oficial con el equipo antes y durante de la competencia. Un equipo debe tener solamente un entrenador, y dicho entrenador no puede ser uno de los competidores.

El entrenador debe registrar el equipo en el sistema de ICPC dentro del tiempo establecido para las competencias regionales. Un equipo no puede ser elegible para competir en el regional si el director regional del ICPC no lo acepta a través del sistema web. Adicionalmente solo los competidores registrados como reserva pueden sustituir a alguno de los competidores, y dichas sustituciones deben ser realizadas por el director regional antes de que comience la competencia.

Como requerimientos básicos cada participante debe cumplir con los siguientes requisitos:

- Un estudiante debe estar disponible para participar en la Final Mundial
- El estudiante debe estar involucrado en estudios superiores con al menos medio tiempo dedicado en la institución.
- Un estudiante solo puede representar a una institución durante un año de competencia.
- Un estudiante que participó en 2 finales mundiales no puede volver a participar.
- Un estudiante que compitió en 5 regionales no puede volver a participar.

Adicionalmente existen restricciones en cuanto al año en que el estudiante comenzó a cursar estudios universitarios y en cuanto a la fecha de nacimiento. Por ejemplo, para participar en la Final Mundial de 2014 es necesario haber comenzado los estudios superiores a partir de 2009 o despues, y haber nacido a partir de 1990. Sin embargo, es posible extender el período de elegibilidad de un competidor cuyos

estudios sean interrumpidos o extendidos, el entrenador debe exponer el caso ante el comité de elegibilidad de ICPC, y recibirá una respuesta, esto debe hacerse 3 semanas antes de la competencia regional. Para visualizar mejor las políticas, se puede consultar el árbol de elegibilidad [5].

1.1.3. Problemas

Como se describe en la sección de problemas de la página oficial de ACM [3], los problemas varían en dificultad y en tipo. Al seleccionar problemas para la competencia, se busca que existan al menos dos problemas que un competidor de segundo año pueda resolver en una hora, dos que puedan ser resueltos por un estudiante de tercer año, y dos que sean los problemas que permitan determinar al equipo campeón. Ningun problema es completamente trivial. El objetivo final es que ningun equipo se quede sin resolver al menos un problema, que cada problema sea resuelto, y ningún equipo pueda resolverlos todos.

Cada problema es presentado en un escenario del mundo real. Los problemarios de las finales mundiales contienen problemas para solucionar planificaciones de trenes, modelar tráfico aéreo, analizar circuitos lógicos, optimizar ubicación de cercas, seguir movimientos de robot, condiciones de competencia, simular recolección de equipaje en aeropuertos, estimar reservas de petróleo, entre otros. Los competidores deben mirar a través de esto, con el objetivo de discernir el problema de fondo y elaborar algoritmos para su solución.

Cada problema consiste en un texto, unos casos de entrada con su respectiva salida. La mayoría de los problemas tienen ilustraciones que ayudan a su comprensión. Los problemas no deben tener más de dos páginas de longitud. Los autores deben asumir que el inglés no es la lengua materna de los competidores, por eso todos la terminología cultural o de la disciplina debe ser explicada en profundidad.

Los competidores recibirán 8 o más problemas en la competencia para ser resueltos en 5 horas. Dichos problemas estarán en lenguaje inglés. Durante la competencia, toda la comunicación oficial entre competidores y jurados será en inglés.

Para resolver los problemas el equipo puede elegir enviar su solución en los lenguajes C/C++, o Java.

1.1.4. Evaluación y Puntuación

Las soluciones enviadas por los competidores a los jurados serán llamadas *runs*. Cada run puede ser aceptado o rechazado, y el equipo recibe la respuesta de los resultados. Cuando una solución es rechazada puede ser por alguna de las siguientes causas:

- run-time error, o error en tiempo de ejecución
- time-limit exceeded, o tiempo límite excedido
- wrong answer, o respuesta incorrecta

La notificación de los *runs* aceptados puede ser suspendida hasta un tiempo apropiado para mantener los resultados finales en secreto. Un anuncio general para tal efecto deberá ser realizado durante la competencia. No obstante, la notificación de los *runs* rechazados se realizará hasta el final de la competencia. Un competidor puede enviar solicitudes de aclaratorias por ambigüedades o errores en el enunciado utilizando un *clarification request*. Si los jurados aceptan que tal solicitud es a lugar, una aclaratoria será enviada a todos los competidores.

Para evaluar las soluciones, los jurados son los únicos responsables en determinar si una solución es correcta o no. Amparado en las decisiones de los jurados, el director de jurados es el responsable de determinar los ganadores de las competencias. Ellos tienen

la autoridad para ajustarse a condiciones especiales o imprevistos, y sus decisiones serán finales.

Los equipos son ordenados de acuerdo al número de problemas resueltos. Los equipos que estén en los primeros 12 lugares que tengan el mismo número de problemas resueltos son ordenados por el tiempo total de manera ascendente, y de ser necesario, por el menor tiempo de envío, del último run realizado aceptado por los jurados. El tiempo total es la suma de los tiempos consumidos para cada problema resuelto. El tiempo consumido para un problema resuelto, es el tiempo que pasa del comienzo de la competencia hasta el primer run que es aceptado para dicho problema, más 20 minutos adicionales por cada intento previamente rechazado para ese problema. No existe tiempo consumido para un problema no resuelto.

1.2. International Olympiad in Informatics (IOI)

La Olimpíada Internacional de Informática IOI, es una competencia internacional en informática, realizada anualmente, entre competidrores de todo el mundo, acompañados por programas sociales y culturales.

La idea de organizar una olimpíada internacional de informática para los estudiantes de escuelas y liceos, fue propuesta en la Conferencia General número 24 de la UNESCO en París, dicha propuesta, fue presentada por el profesor Búlgaro Blagovest Sendov, en octubre de 1987. Este plan incluído en el 5to programa principal de la UNESCO para el bienio 1988-1989. En mayo de 1989, la UNESCO inició y fue el patrocinante de la primera Olimpíada Internacional de Informática, realizada en Bulgaria en 1989, como se expresa en la sección de historia de su página oficial [7].

Esta Olimpíada es una de las 5 olimpíadas internacionales en ciencias. El objetivo primario del evento, es estimular a los estudiantes en las ciencias de la computación, y tecnologías de información. Otro objetivo importante es reunir a los estudiantes

más destacados de varios países del mundo y ofrecerles la oportunidad de compartir experiencias científicas y culturales.

En las secciones a continuación se presentan las reglas de la competencia, condiciones para participar, descripción de los problemas y su evaluación. Todas esta normativa fue aprobada en el año 2010, en la reunión de la Asamblea General durante la IOI de Canadá [8].

1.2.1. Formato de Competencia

La competición toma lugar en 2 días de competencia, los cuales están intercalados con días libres de competencia. El país organizador deberá proveer un día de práctica previo a los días de competencia para que los competidores puedan probar el equipo y el ambiente de trabajo.

En los días de competencia los competidores deberán resolver los problemas propuestos, para ello contarán solamente con la versión traducida en su lengua materna, así como el material permitido por los procedimientos de competencia. No puede existir comunicación entre los competidores, más allá de lo que pueda permitirse en los procedimientos de competencia.

El país organizador es el encargado de asegurar que los líderes y acompañantes no tengan contacto con los competidores una vez revelados los problemas en la asamblea general para su discusión, hasta culminado el día de competencia. Ningún tipo de comunicación puede existir entre líderes o acompañantes durante este período.

Durante la primera mitad de cada día de competencia, los líderes de las delegaciones deben estar presentes, para de ser necesraio traducir al inglés las preguntas realizadas por los competidores acerca de los problemas. Las preguntas deben ser respondidas por el comité científico, y de ser requerido por la asamblea general, con una frase *YES, NO* o *NO COMMENT*.

El comité científico estará encargado de presentar las reglas de competición, procedimientos de evaluación y los problemas que se van a utilizar en los días de competencia, en las reuniones previas a cada día realizadas por la asamblea general. Durante ese momento serán votados los problemas a utilizar, y luego de ser aprobados, los líderes y acompañantes serán los encargados de realizar las traducciones para los estudiantes manteniendo el apego estricto al original.

1.2.2. Participantes

Cada país participante es representado por una Delegación Nacional, todos los miembros de la delegación representan a una nación. Cada delegación nacional estará conformada por un Líder de Delegación, junto a un equipo de hasta 4 competidores, y si hay más de un competidor, un Líder delegado.

El líder delegado puede actuar en representación del Líder de la Delegación en cualquier situación, así como asesorarlo en sus tareas. Para los líderes es necesario tener el conocimiento técnico, que permita asesorar a los competidores en los problemas que puedan surgir durante las competencias.

Un competidor es un estudiante que fue admitido en una escuela de educación media, en el país al cual representa, durante el período de Septiembre a Diciembre del año anterior a la IOI, y no es mayor de 20 años para el 1 de julio del año de la IOI. Los estudiantes que estén estudiando en el extranjero pueden representar al país de su nacionalidad.

Cabe destacar que para la IOI, un **país** es un estado que ha sido oficialmente reconocido por las Naciones Unidas, la UNESCO, o cualquier país que haya formado parte de la competencia anteriormente. La responsabilidad de organizar el evento corresponde a un país sede, generalmente conformado por ministerios, instituciones, y corporaciones en dicho país.

1.2.3. Problemas

La descripción de los tipos de problemas utilizados en la IOI está contenida en el temario IOI [6]. El primer propósito de dicho documento es proveer una serie de lineamientos para ayudar a decidir cuándo un problema es apto para ser utilizado en la IOI. Basado en este documento, el Comité Científico Internacional (ISC por sus siglas en inglés), evalua los problemas propuestos para ser utilizados en la IOI. El segundo propósito del temario, es proveer a los estudiantes una guía de tópicos a preparar para participar en la IOI.

Para lograr estos objetivos, el temario de la IOI, provee una clasificación de tópicos y otros conceptos desde matemáticas hasta ciencias de la computación. En particular, algunos de los tópicos sencillos son clasificados como *incluídos*, así como por otra parte, tópicos difíciles son *excluídos* de forma explícita. De manera más precisa, el documento clasifica los tópicos en 5 categorías:

- **Incluído, ilimitado**: los tópicos en esta categoría son considerados como conocimiento requerido para participar. Se espera que los competidores dominen el contenido. Estos tópicos pueden aparecer en los enunciados, sin requerir mayor explicación. Por ejemplo: *Integer*
- **Incluído, a ser aclarado**: los competidores deben conocer estos tópicos, pero cuando aparezca en un enunciado, el autor debe aclararlo siempre de manera suficiente. Por ejemplo: *grafo dirigido*.
- **Incluído, no para la descripción del enunciado**: los tópicos dentro de esta categoría no deben aparecer en los enunciados, sin embargo, desarrollar soluciones, y entender como modelar la solución puede requerir el uso de estos tópicos. Por ejemplo: *Análisis asintótico para determinar un límite superior de complejidad*.

- **Fuera de foco**: esta es la nueva categoría por defecto. Cualquier tópico no mencionado en el temario debe ser considerado dentro de esta categoría. No se espera que los competidores tengan estos conocimientos. La mayoría de los problemas no estarán relacionados con tópicos dentro de esta categoría. Por ejemplo: los problemas *Lenguajes* (también conocido como Wikipedia) utilizado en la IOI 2010 en Canadá, y *Odómetro* (también conocido como robot with pebbles) de la IOI 2012 en Italia.

  Sin embargo, no es la intención del temario evitar la inclusión de algún tópico dentro de esta categoría en algun problema, el ISC puede desear incluir un problema de este tipo para ampliar el ámbito de los problemas de la IOI. Si algún problema relacionado con un tópico de esta categoría es considerado para la IOI, el ISC deberá asegurar que dicho problema puede ser resuelto sin conocimiento previo del tópico, y puede ser presentada la tarea en términos de tópicos incluídos, e incluídos para ser aclarados, de forma precisa y concisa.

- **Explícitamente excluído**: algunos de los tópicos algorítmicos más difíciles están explícitamente exluídos de la IOI, y está garantizado que ningun problema requerirá que los competidores tengan conocimiento de esos tópicos. Esta categoría contiene principalmente algoritmos difíciles, que se escapan del alcance de la IOI. Por ejemplo: *Algoritmos de Maximum flow*.

Cabe destacar que lo expresado en el temario no debe ser interpretado para restringir de ninguna manera, las técnicas que los competidores tienen permitido utilizar para solucionar los problemas.

Entre los tópicos incluídos dentro del temario tenemos los siguientes: operaciones aritméticas y geométricas, conjuntos y relaciones, funciones, lógica básica, inducción y pruebas matemáticas, conteo básico, grafos y árboles, recursión, estrategias de divide y conquista, backtracking recursivo, algoritmos de fuerza bruta, algoritmos avaros, búsquedas y manipulación de strings, programación dinámica, algoritmos simples de

teoría de números, algoritmo de Euclides, la criba de Eratóstenes, factorización, exponenciación eficiente y ordenamiento entre otros. Adicionalmente se asumen los conocimientos básicos para el manejo de los lenguajes de programación, conocimiento del uso de estructuras de datos y funcionalidades provistas por los lenguajes, habilidades de depuración de código, eficiencia, así como utilización de librerías particulares para algún problema.

1.2.4. Evaluación y Puntuación

Como se mencionó anteriormente, en cada olimpíada será presentado el procedimiento para evaluar los problemas por parte del comité científico, no obstante, se mencionan algunos lineamientos generales para la evaluación y puntuación durante el evento.

Luego de realizar la evaluación de cada solución enviada por los competidores, el comité científico entrega los resultados de cada una, al líder de la delegación para su evaluación, en conjunto con los competidores. De ser necesaria alguna apelación, el líder será el encargado de presentarla a algún miembro del comité científico, y de ser necesario, será evaluado por todo el comité, o inclusive por el comité internacional y la asamblea general.

La evaluación de los problemas arrojará un puntaje para los competidores, y dicho puntaje determinará el resultado de la competencia. Los miembros de la asamblea general deberán confirmar los resultados de los competidores, con el objetivo de entregar las medallas según el siguiente orden:

- No más de la mitad de los competidores reciben medallas
- Una doceava parte de los competidores recibe medallas de oro
- Un sexto de los competidores recibe medallas de plata

- Un cuarto de los competidores recibe medalla de bronce

En las últimas ediciones de la IOI (desde el 2010), la evaluación de los problemas, con algunas variaciones, ha sido estructurada según el siguiente esquema:

- Cada problema tiene un conjunto de casos de prueba para su evaluación.
- Los casos de prueba están agrupados en subtareas, las cuales, tienen un puntaje asignado dependiendo de su dificultad
- Para obtener el puntaje asociado a una subtarea, la solución del competidor, debe resolver todos los casos de prueba contenidos en una subtarea.
- El puntaje final para un problema será igual a la suma de todas las soluciones para un problema.
- Si el competidor suministró varias soluciones, será tomada en cuenta aquella que obtenga la mayor cantidad de puntos.

1.3. Competencias por internet

Con el auge de Internet, también crecieron las competencias de programación por Internet, tanto así, que hay lugares donde se realizan competencias de un nivel muy alto, donde compiten los mejores a nivel mundial, generalmente esquemas de gamificación,[1] combinados en muchos casos con premios cuantiosos en efectivo, además de competencias presenciales.

En las siguientes secciones se mencionarán las competencias de programación más populares por Internet.

[1]Gamificación es el uso de la forma de pensar y trabajar en los juegos, pero aplicado a otras áreas con el fin de atraer a usuarios para solucionar problemas

1.3.1. TopCoder Algorithm

TopCoder es una comunidad web de desarrolladores donde se realizan diversos tipos de competencias, entre ellas las competencias de algoritmos o *Algorithm Competitions*.

Dentro de las competencias de algoritmos existen dos tipos de eventos, *Single Round Matches* o SRM por sus siglas en inglés, que se realizan de manera regular, y está el torneo por eliminación que se realiza anualmente conocido como *TopCoder Open* o TCO por sus siglas en inglés.

Cada ronda de competencia en la cual participe un concursante, afectará directamente su *rating*, sin importar que esté participando en un SRM o una ronda del TCO. Las variaciones del rating serán explicadas con mayor detalle más adelante.

Todos los detalles de la participación en la competencia, y como es el desarrollo de la misma son descritos a continuación y se expresan en el instructivo para competencias de programación de TopCoder[9]. Para registrarse y utilizar el sistema de competencias en TopCoder, puede visitarse la referencia de *Como competir en los SRM's de TopCoder* [11].

1.3.1.1. Formato de Competencia

Todas las fases de una ronda de competencia en TopCoder deben comenzar y terminar al mismo tiempo para cada competidor. Las funcionalidades en la arena van a variar dependiendo de la fase de competencia que se este realizando. Cada ronda de competencia sea de un SRM o una ronda particular del TCO estará formada por las siguientes fases:

- Fase de Codificación: la fase de codificación es el período durante el cual cada competidor intenta realizar las soluciones a los tres problemas en los enunciados.

En la mayoría de los casos, la fase de codificación tendrá como duración 75 minutos, dicho tiempo es el tiempo total que puede utilizar el competidor para entregar las soluciones que tenga para cualquiera de los problemas.

Cuando la fase de codificación comienza, cada competidor tiene la oportunidad de ver los enunciados. Los problemas poseen una puntuación, mientras mayor sea la cantidad de puntos indicada para el problema, mayor será su dificultad. Los competidores pueden abrir los problemas en cualquier orden que consideren, y tan pronto como un problema es seleccionado, los puntos que otorga el problema comenzaran a decrementarse. Mientras mayor sea la cantidad de tiempo que un problema está abierto, menor será la cantidad de puntos que pueda obtener un competidor. Los competidores también tienen la opción de abrir múltiples problemas al mismo tiempo, pero ello hará que se decremente la puntuación para todos los problemas abieros. Cerrar un problema, no causará que se dentenga el proceso de decrementar el valor de dicho problema.

Una vez abierto el problema, al competidor se le presentará una ventana de codificación. Dicha ventana de codificación contiene las herramientas necesarias para realizar la solución del problema, compilar la solución, probar la solución, y enviar la solución.

- Fase de retos o Challenge: la fase de retos generalmente comienza 5 minutos luego de terminada la fase anterior, y dura 15 minutos. Durante esta fase, los competidores tienen la oportunidad de ver el código fuente de las soluciones enviadas por los otros competidores en la misma sala.

  Si un competidor considera que cualquiera de las soluciones de los otros competidores tienen fallas, dicho competidor puede retar la solución con un caso de prueba específico, en el cual, el competidor crea que la solución en cuestión retornará un resultado errado. Si el reto está en lo cierto, y la solución falla, el competidor que hace el reto recibirá 50 puntos y el competidor dueño de la solución perderá todos los puntos recibidos por esa solución. Sin embargo, si el

reto está equivocado y la solución retorna un resultado correcto, el competidor que hace el reto perderá 25 puntos de su puntuación total.

Ciertas restricciones aplican durante la fase de retos, dichas restricciones son las siguientes: Un competidor sólo puede intentar un reto si posee un puntaje mayor o igual a 0, si el reto va dirigido a una solución de un competidor en su misma sala, la solución no ha sido objeto de un reto satisfactorio anteriormente, y no es una solución del mismo competidor.

Los casos de prueba utilizados en los retos que resulten satisfactorios, seran incluídos en la prueba del sistema.

- Fase de Pruebas del Sistema o System Test: la fase de pruebas del sistema, es una fase no interactiva. Inmediatamente despues de la fase de retos, los servidores de TopCoder ejecutarán una serie de pruebas. Cada solución que se mantenga hasta esta fase será sujeta a una serie de pruebas extensas.

  El sistema se asegurará de que cada solución enviada, retorne un resultado correcto, utilizando a lo sumo 2 segundos por cada caso de prueba. Si alguna solución falla alguno de los casos de prueba, esa solución será marcada como incorrecta y los puntos obtenidos por esa solución seran eliminados del puntaje total del competidor. Esta fase toma normalmente entre 10 y 20 minutos, algunas veces un poco mas.

1.3.1.2. Participantes

Para ingresar a la arena de TopCoder, el competidor deberá estar registrado como un miembro de TopCoder. Para hacerlo el competidor deberá completar un formulario en el sitio web de TopCoder. Una vez registrado, el competidor tendrá acceso a la arena de competencia.

Para participar en una ronda de competencia, los competidores deben ingresar a la arena de competencia, y registrarse en la ronda. Para registrarse en la ronda de

competencia, en la sección de *Active Contests* estarán los enlaces correspondientes para el registro.

Una vez finalizado el período de registro para una ronda (5 minutos antes de comenzar), se realiza la *asignación de salas*. El resultado de ello, es que cada competidor será asignado a una sala en la cual realizará su competencia. La asignación de salas tiene relevancia en la fase de challenges o retos, dado que solo es posible hacer retos a los competidores en su sala.

1.3.1.3. Problemas

En TopCoder, existen dos divisiones de competencia, la división específica en la cual deba competir un participante va a depender únicamente en su *rating*. Los competidores con un rating de 1200 puntos en adelante deberán competir en la división 1, de cualquier otra manera, los competidores deberán participar en la división 2. Luego de cada competencia se actualizan los ratings, para determinar las divisiones en las cuales competiran los participantes.

Los problemas en la división 1 son más difíciles que los problemas de la división 2. Con frecuencia, un problema de la división 1 es compartido con la división 2, generalmente es el problema de mayor puntaje en la división 2.

El ámbito de los problemas está centrado en su naturaleza algorítmica, los problemas están diseñados para que la solución sea procesar algunos casos de prueba como entrada, y producir una salida. Abarcan desde problemas aritméticos, lógicos, geométricos, hasta algoritmos de flujo en redes, programación dinámica y manejo de grafos, envueltos en enunciados que los enlazan con problemas de la vida real.

Quedan fuera del alcance de las competencia problemas que impliquen otro tipo de conocimiento no relacionado con la algorítmica, como por ejemplo: realización de interfaces gráficas, manejo de sistemas operativos o de bases de datos, sistemas distribuidos, etc.

1.3.1.4. Evaluación y Puntuación

La evaluación y el puntaje de un problema va a depender del nivel de dificultad del problema y el tiempo que tomó al competidor elaborar y enviar la solución. Para ello los servidores de TopCoder van a calcular el tiempo empleado en solucionar el problema, contando el tiempo a partir del cual el competidor abre el problema hasta que entrega la solución. Debe tomarse en cuenta que las capturas de tiempo son realizadas cuando se producen las peticiones al servidor de TopCoder, la latencia de la red no es tomada en cuenta por TopCoder.

El puntaje total obtenido por un problema se calcula mediante la siguiente fórmula:

$$Puntajetotal = MP * (0{,}3 + \frac{0{,}7 * TT^2}{10 * PT^2 + TT^2}) \tag{1.1}$$

Donde PT es el tiempo empleado en la elaboración de la solución del problema, TT es el tiempo total empleado para solucionar todos los problemas, y MP es el máximo de puntos disponibles por ese problema.

Al concluir la fase de las pruebas del sistema, TopCoder realizará el cálculo de los ratings de todos los participantes en la ronda de competencia. Si un competidor no abre ningún problema, su rating no será modificado, y una vez completado el cálculo será notificado por los administradores a través de un mensaje en la arena.

El cambio en el rating expresa una medida de *que tan bien ha sido evaluado un competidor en comparación a los otros en su división*, dado *que tan bien se esperaba que saliera dicho competidor basado en su rating*. El rating aumenta, cuando el resultado excede las expectativas. Para una explicación mas detallada del rating, se puede ver la sección del sistema de rating para las competencias de programación [10].

1.3.2. Codeforces

Codeforces es un proyecto que permite unir a las personas interesadas en participar en competencias de programación, según se expresa en su sección de preguntas frecuentes [12]. Por una parte, Codeforces funciona como una red social dedicada a la programación y a las competencias de programación. Y por otra parte, es una plataforma donde se realizan competencias de manera regular, donde las habilidades de los participantes son medidas a través del rating. Codeforces se encuentra en un desarrollo constante, en aras de permitir mejoras en la plataforma que permita a los participantes la oportunidad de organizar sus propias competencias, llenar el proyecto con conocimiento educativo, y desarrollar Codeforces como una plataforma de entrenamiento.

En las siguientes secciones se describe el funcionamiento de las competencias dentro de la plataforma de Codeforces según lo descrito en su página de reglas [13].

1.3.2.1. Formato de Competencia

Una ronda de competencia dura generalmente 2 horas, en las cuales, se presentan a los competidores 5 problemas para resolver, a menos que se indique lo contrario. Los problemas estarán disponibles inmediatamente para los competidores al momento de comenzar la ronda, en dos lenguajes, inglés y ruso.

Momentos antes de comenzar la ronda, todos los competidores serán divididos en salas, cada sala puede contener cerca de 40 competidores.

Durante la competencia, las soluciones presentadas por los competidores serán evaluadas con un número reducido de casos de prueba, llamados *pretests*. El competidor recibirá el resultado de los pretests para cada solución justo despues de enviarla. Cada caso de prueba deberá apegarse a los límites para las variables, el tiempo y la memoria por caso de prueba, indicados en el enunciado del problema.

El caso de prueba será considerado bueno, si el programa del competidor produce la respuesta esperada para los casos de prueba, retorna como exitcode 0, dentro de los límites de memoria y tiempo establecidos.

Una de las variantes importantes con respecto a otras competencias son los Hacks de Codeforces. Un competidor puede bloquear alguno de los problemas, para el cual, el competidor haya solucionado correctamente los pretests, esto implica que el competidor renuncia a su derecho de enviar más soluciones para dicho problema.

Luego de bloquear un problema, el competidor obtiene el derecho de ver las soluciones de otros competidores para ese mismo problema, dentro de su sala. Habiendo hecho esto, el competidor puede proponer un caso de prueba, en el cual, piense que el código que esta viendo pueda fallar. Este procedimiento es lo que se conoce como *hack*.

Un caso de prueba puede ser escrito de forma manual, o utilizando un programa generador, que escriba el caso de prueba por la salida estándar. El sistema automáticamente validará que el caso satisface las condiciones del problema, si no lo hace, el competidor es notificado con que su intento de hack ha sido ignorado. Si el caso de prueba satisface las restricciones del problema, la solución se ofrece como prueba al programa del competidor víctima del hack. Si la solución falla el hack, el intento de hack es considerado exitoso, de otro modo es considerado fallido.

Un intento de hack puede ser ignorado también si para el momento que se intenta hacer el hack, la solución ya fue hackeada por otro competidor anteriormente.

Los competidores que realicen hacks exitosos recibirán 100 puntos, sin embargo, un intento de hack fallido le costara 50 puntos al competidor que lo intente.

Una vez hackeada una solución, sucede lo siguiente: el problema no se considera pre-resuelto por el competidor víctima del hack, así como sus puntos para esa solución regresan a 0, y el caso de prueba utilizado en el hack es añadido a las pruebas del sistema.

Si un competidor bloquea un problema, y resulta su solución hackeada, preserva el derecho para realizar hacks a otros competidores en ese mismo problema.

Luego de culminada la competencia, tiene lugar una fase donde se realizan la evaluación de las solucines con los casos de prueba definitivos, para todas las soluciones que aprobaron los pretests y no han sido hackeadas enviadas por los competidores. En base a esta fase final se calculan los resultados definitivos para la competencia.

1.3.2.2. Participantes

La participación en los eventos es gratuita y abierta a todo el que quiera competir. Para hacerlo, el competidor debe completar un registro en el sitio y registarse para participar en cada ronda de competencia. El registro comienza 6 horas antes de comenzar la ronda y cierrra 5 minutos antes de comenzar.

1.3.2.3. Problemas

Los problemas tienen una naturaleza algorítmica, diseñados para evaluar los conocimientos de los competidores en algoritmos y ciencias de la computación. La restricción es que deben ser problemas a los cuales se proporcione una entrada, generalmente por la entrada estándar y produzcan una salida generalmente utilizando la salida estándar.

Los problemas son redactados enmarcando situaciones del mundo real, o situaciones de fantasía a discreción del autor, sin embargo, la solución adyacente al problema está conformada por algoritmos y métodos clásicos de programación.

1.3.2.4. Evaluación y Puntuación

Como se mencionó anteriormente las soluciones a los problemas son evaluadas inicialmente con un número reducido de casos de prueba denominados pretests. Du-

rante esa evaluación, de resultar fallida su solución, el competidor recibirá alguno de los siguientes mensajes: Memory limit exceeded, Time limit exceeded, Runtime error, Wrong answer, Idleness limit exceeded (el programa no utilizó la CPU por un período de tiempo considerable), Denial of judgement (la solución es imposible de probar debido a errores durante la evaluación, posiblemente causados por el programa del competidor).

Si una solución aprueba los pretests, el compoetidor obtiene un resultado que dice **Pretests passed**, de cualquier otra forma, el competidor recibe el resultado de su evaluación como el número del pretest que su solución falla.

Si el resultado de una solución arroja **Compilation error**, o si la solución falla el primer pretest no será considerada para los cálculos finales de los resultados.

Una solución para un problema puede ser enviada múltiples veces.

Una solución que logre aprobar todos los pretests, será considerada como la solución **verificada** por el competidor. Si el competidor envía varias soluciones consideradas como verificadas, entonces sólo la última será considerada como verificada, y todas las demás serán soluciones fallidas.

La puntuación para un problema luego de que el competidor logra resolver los pretests se realiza de la siguiente manera:

- Cada minuto se decrementa el valor del problema, el valor disminuye con un factor de $X/250$ puntos por minuto (donde X es el valor inicial del problema), por ejemplo, un problema con un valor de 500 puntos, pierde 2 puntos por minuto.
- El número de puntos que un competidor obtiene por problema es equivalente al valor actual de los puntos del problema, menos los puntos de penalización.
- El tiempo de penalización es determinado por el número de soluciones enviadas por el competidor previamente para ese problema, múltiplicado por 50 puntos.

- Finalmente un competidor no puede obtener para un problema, menos del 30 % de los puntos que inicialmente valía el problema.

Luego de finalizada una competencia, el resultado final corresponde a los puntos obtenidos por el competidor en cada solución que no sea hackeada, y pase todas las pruebas del sistema incluyendo los pretests, más el puntaje obtenido por los hacks. Los competidores serán ordenados en una tabla ordenada descendentemente por puntaje, y si dos o más competidores comparten la misma cantidad de puntos, compartirán la misma casilla.

1.3.3. Google Code Jam

El Google Code Jam es una competencia diseñada para involucrar a los programadores de todo el mundo en competencias de programación de naturaleza algorítmica. Ofreciendo premios para aquellos competidores más destacados durante las diferentes rondas de la competencia. Toda la descripción del funcionamiento, reglas y condiciones del evento se describen en el acuerdo de términos y condiciones presentado en su página web [15], adicionalmente se provee una guía rápida para su entendimiento en la sección de prequntas frecuentes [14].

1.3.3.1. Formato de Competencia

El Google Code Jam se divide en múltiples *Rondas*, períodos de tiempo fijos en los cuales los competidores deben resolver un conjunto de problemas algorítmicos. Las rondas toman lugar en horarios definidos. Los competidores que lo hagan lo suficientemente bien clasifican para la siguiente ronda, hasta la ronda final, en la cual, resulta un campeón. Las rondas son de diferente duración, variando de 2 a 4 horas, exceptuando a la ronda de calificación que dura 25 horas.

Desde el 2012 la estructura de las rondas en el Google Code Jam funciona de la siguiente manera:

- Ronda de Calificación: esta es la primera ronda y dura 25 horas. No es necesario estar listo para competir al momento de iniciar la ronda, está dispuesto que dure 25 horas para que los competidores alrededor de todo el mundo puedan participar en el momento más conveniente para ellos. Es recomendable apartar 2 horas dentro de las 25 para competir. Cualquier competidor capaz de obtener un número fijo de puntos, avanzará a la siguiente ronda. La tabla de puntuación estará disponible para los competidores en el sitio web de la competencia.

- Ronda 1: para esta ronda existen 3 sub rondas, la ronda **1A**, ronda **1B** y ronda **1C**. Cada una tiene una duración de dos horas y treinta minutos. Un competidor puede participar en las 3 rondas si lo desea, pero al quedar entre los primeros 1000 competidores en alguna de las rondas, automáticamente quedará clasificado para la ronda 2, y no le será permitido participar en las rondas restantes de la Ronda 1.

- Ronda 2: los 3000 competidores que avanzan al terminar la Ronda 1, participan en esta ronda. Esta ronda durará 2 horas y 30 minutos. Los primeros 1000 competidores ganarán una camisa, y los 500 primeros competidores avanzan a la ronda 3.

- Ronda 3: los 500 competidores que avanzaron luego de la ronda 2, compiten en esta ronda. Dicha ronda tiene una duración de 2 horas y media. Los primeros 25 competidores avanzan a la final. Si alguno de esos competidores son inelegibles para ir a las finales, Google puede seleccionar a los siguientes competidores según el ranking para completar el ranking.

- Final: los 25 competidores seleccionados para ir a la Final, deben viajar a la oficina Google designada para el evento y competir por el premio final, y el título de campeón del Google Code Jam.

1.3.3.2. Participantes

Para participar en el Google Code Jam, los competidores deben registrarse en el sitio web de la competencia durante la fecha de registro. Dicha fecha está publicada en el sitio, en un calendario de dominio público.

Para participar un competidor debe tener 13 años o más, y para poder participar en la final los competidores deben tener 18 años o más. Adicionalmente los competidores no pueden ser empleados o pasantes dentro de Google Inc. o alguna empresa afiliada o subsidiaria de Google. Si el competidor tiene una oferta de trabajo de Google, debe detener su participación al momento de convertirse en empleado. Cabe destacar que de ser elegible un competidor para participar en la final, y concretarse alguna oferta de trabajo con Google, dicho competidor no será elegible para participar en la final.

Tampoco son elegibles para participar competidores familiares inmediatos o residentes en el mismo lugar de empleados de Google Inc, o empresas afiliadas o subsidiarias, así como residentes de Quebec, Arabia Saudita, Cuba o Syria, o en cualquier otro lugar que prohiban las leyes norteamericanas.

1.3.3.3. Problemas

Durante cada ronda de la competencia, serán presentados a los competidores de cada ronda un conjunto de problemas para su resolución. Cada conjunto de problemas estará compuesto de una serie de tareas de naturaleza algorítmica, en conjunto con las entradas y salidas para cada problema como se describe a continuación. Una vez que una ronda comienza, cada competidor tendrá acceso a los problemas y podrá descargar los archivos de entrada para los problemas en esa ronda. Adicionalmente a la descripción del problema, cada uno tendrá asociado uno o más conjuntos de entradas/salidas, las cuales pueden incluir uno o más conjuntos de entrada/salida para los casos pequeños, entrada/salida para los casos grandes, u otro tipo de entradas/salidas descritos en el problema.

Existen además diferentes reglas para los casos de prueba pequeños y los casos de rueba grandes:

- Casos de prueba pequeños: cuando un competidor intenta resolver un conjunto de casos de prueba pequeño, un contador de tiempo comienza tan pronto como presiona el competidor el botón de descargar los archivos de entrada. El competidor tendrá 4 minutos para enviar el archivo salida correspondiente a la entrada recibida, así como el código fuente utilizado para generar dicha salida. Si el archivo de salida y el código fuente no son recibidos dentro de los 4 minutos, o si la salida generada está incorrecta, el intento sera evaluado como incorrecto, y el competidor será notificado de inmediato. En este punto, el competidor puede optar por intentar nuevamente el problema, pero deberá descargar un nuevo archivo de entrada. Los envíos realizados por los competidores, serán evaluados de manera inmediata, notificando a los competidores con el resultado si es correcto o incorrecto. Para algunos tipos de envíos incorrectos, el intento puede ser ignorado. El competidor será notificado si su salida está mal formada. Adicionalmente el competidor puede enviar tantas veces como quiera sus soluciones durante el lapso de los 4 minutos.

- Casos de prueba grandes: un competidor debe solucionar de forma correcta los casos de prueba pequeños antes de intentar los casos de prueba grandes, a menos que se especifique otra cosa en el enunciado del problema. Cuando un competidor intenta resolver los casos de prueba grandes, comienza un contador de tiempo tan pronto como presiona el botón para descargar el archivo de entrada. El competidor tendrá 8 minutos para enviar el archivo de salida correspondiente, así como el código fuente utilizado para generar dicha solución. Cada competidor tiene permitido descargar un archivo de entrada grande por problema. El competidor puede enviar múltiples salidas junto a su código fuente durante el período de los 8 minutos, sin embargo, sólo será tomado en cuenta el último envío realizado. Los resultados de la evaluación para los casos de prueba grandes, serán

revelados a los competidores luego de terminada la ronda de competencia. Para cierto tipo de soluciones incorrectas, el intento puede ser ignorado. El competidor será notificado si su salida está mal formada. Adicionalmente el competidor puede reenviar sus soluciones en cualquier momento durante los 8 minutos, pero no está permitido hacer nuevos intentos para los casos de prueba grandes luego de consumir los 8 minutos del primer intento.

- Otros conjuntos de entrada y salida: algunos problemas pueden utilizar otros tipos de entradas y salidas, diferentes a los casos pequeños y a los casos grandes. Los enunciados para dichos problemas deben expresar las condiciones para dichos casos.

Es recomendado a los competidores enviar sus soluciones con una cantidad de tiempo restante adecuada para evitar problemas derivados de la latencia de la red, así como inconvenientes que se puedan presentar con los servidores de Google.

Para considerar un intento de solución de un problema válido, este debe contener la salida generada por el programa del competidor para la entrada descargada, más todo el código fuente necesrio para generar dicho archivo de salida. Los envíos deben ser hechos dentro del tiempo determinado para ser considerados. Adicionalmente los archivos de salida deben estar en el formato dispuesto en el sitio de competencia.

Para subir los archivos los competidores pueden hacerlo mediante archivos planos o comprimidos. El tamaño máximo para los archivos de salida es de 100KB, y el tamaño total de todos los archivos fuentes no deben exceder 1MB. El código deliberadamente ofuscado no está permitido.

Durante cualquier ronda de la competencia, si algún competidor requiere notificar o preguntar algo a los jueces, puede hacerlo a través del enlace **Ask a Question**. Del mismo modo si el competidor considera que envió de manera incorrecta un código fuente para una solución, los jueces pueden decidir si marcar la solución como incorrecta y permitir al competidor intentar el problema nuevamente. En el caso de la

ronda final solamente, si un competidor decide que el código que ha enviado para la entrada pequeña o grande es incorrecta, el competidor puede solicitar a través del enlace **Ask a Question**, y solicitar el reenvío de del código fuente solo para ese conjunto de casos de prueba. Los jurados tendrán la potestad de permitir el reenvío del código, mas no así el archivo de salida enviado por el competidor. Luego de finalizada una ronda los competidores no pueden reportar haber enviado el código fuente para solicitar reenviarlo.

1.3.3.4. Evaluación y Puntuación

Todos los archivos enviados serán evaluados de la misma forma utilizando el siguiente esquema de puntuación:

Cada problema tiene puntajes fijos para sus conjuntos de entradas y salidas. Por ejemplo, para un problema con casos de prueba pequeños y casos grandes, los casos pequeños pueden valer 10 puntos, mientras la solución de los casos de prueba grandes pueden valer 15 puntos. El puntaje total de un competidor para una ronda, será la suma de todos los puntos obtenidos para los conjuntos de casos de prueba que se resuelvan de manera correcta, de forma independiente entre cada conjunto de casos de prueba.

En el caso de que dos o más competidores queden empatados en puntuación, dichos competidores serán ordenados de manera ascendente según el tiempo acumulado para lograr sus soluciones. En otras palabras, mientras los competidores estén empatados en puntos, el que tenga un menor tiempo acumulado en sus soluciones será primero en el orden, mientras que el que tenga mayor tiempo acumulado será ubicado último en el orden.

El tiempo de un competidor para una ronda determinada, es igual al momento en el cual, el competidor envió su última solución evaluada correcta por los jurados (esto medido desde el inicio de la competencia), más 4 minutos adicionales por cada

intento incorrecto para solucionar un caso pequeño de un problema, donde el competidor finalmente resuelve correctamente el caso pequeño al menos. Los problemas que no tengan las reglas de los casos de prueba pequeños y grandes, especificaran las condiciones específicas en el enunciado.

Si al final de una ronda, es detectada alguna discrepancia entre el código fuente de un competidor y los archivos de salida proporcionados, donde la solución fue juzgada de forma correcta por los jurados, se conformará un panel de jurados y empleados de Google o empresas afiliadas. Los jurados pueden determinar si existe tal discrepancia, y decidir si es una discrepancia menor o mayor, si es menor, el competidor recibirá una penalización de 4 minutos para ese conjunto de casos de prueba. En el caso de una discrepancia mayor el competidor perderá todos los puntos para ese conjunto de casos de prueba, y si el jurado no juzga que no hay tal discrepancia, no se realizará ningún cambio en la puntuación y ninguna penalización será aplicada.

1.3.4. Facebook Hacker Cup

La Facebook Hacker Cup es una competencia de programación mundial, realizada anualmente, donde "Hackers" compiten entre sí para buscar un campeón. En las siguientes secciones se describe a fondo el funcionamiento de la competencia. Al igual que en Google CodeJam, el Facebook Hacker Cup presenta una descripción detallada de la competencia, junto con las condiciones y reglas de participación [16], así como una guía rápida de participación [17].

1.3.4.1. Formato de Competencia

La competición está dividida en múltiples rondas, en cada una de ellas, los competidores reciben su puntuación basados en las soluciones propuestas para una serie de problemas, como se describe más adelante. Luego de cada ronda, un número específico de competidores, con los puntajes más altos en esa ronda avanzan a la siguiente ronda.

Los competidores deben solucionar de forma satisfactoria al menos un problema para avanzar a la siguiente ronda de la competición. La ronda de calificación y las rondas 1, 2, y 3 serán realizadas de forma online, sin embargo, la ronda final será presencial en el lugar dispuesto por Facebook.

- Ronda de calificación: la competición comienza con una ronda de calificación de 72 horas, a todos los competidores registrados se les presentaran 3 problemas, cada uno de ellos consiste en un archivo con la entrada. Los competidores pueden entrar al sitio de competencia en cualquier momento durante las 72 horas e intentar resolver los problemas, sólo los competidores que logren resolver un problema satisfactoriamente avanzan a la ronda 1.

- Ronda 1: La ronda 1 tiene una duración de 24 horas, durante ese período, los competirodes deben ingresar al sistema y les serán presentados una serie de problemas descritos más adelante. Los primeros 500 competidores avanzan a la segunda ronda. Cualquiera que obtenga la misma cantidad de puntos que tiene la persona en el puesto 500 también avanzará a la ronda 2.

- Ronda 2: los competidores que superaron la ronda 1, se enfrentaran en una ronda con una duración de 3 horas, para ello, los competidores deben ingresar al sitio de la competencia. En ese sitio se le presentará a cada competidor un conjunto de problemas, para ser resuelto y explicado como se resume en la siguiente sección. Los primeros 100 competidores de esta ronda obtienen una franela oficial de la competición.

- Ronda 3: los primeros 100 competidores que avancen a esta ronda, recibiran un correo electrónico al terminar la ronda 2. La ronda 3 tendrá una duración de 3 horas. En dicha ronda los competidores deberán ingresar al sistema de competencia durante las 3 horas, allí se presentaran los problemarios para ser resueltos como se indica más adelante. Los primeros 25 competidores de esta ronda serán notificados por email para asistir a la final presencial.

- Ronda Final: los primeros 25 competidores seleccionados avanzan a esta ronda final. Durante la ronda final, los competidores deben resolver un conjunto de problemas, utilizando sólo el equipo provisto por Facebook para ello, otro tipo de material puede ser o no permitido por el personal de Facebook. Adicionalmente los competidores serán notificados con los premios para cada puesto disputado en la final.

1.3.4.2. Participantes

La competición está abiera para aquellos individuos que estén registrados en Facebook, estén de acuerdo con los términos y condiciones presentados en el sitio web de la competición y que sean mayores de 18 años. Los empleados actuales, pasantes, contratistas de Facebook o alguna de las empresas subsidiarias de Facebook, así como sus familiares directos y miembros de las empresas patrocinantes del evento, no son elegibles para participar en la competencia. Las personas que vivan en países como Quebec, Arabia Saudita, Cuba o Syria, o en cualquier otro lugar que prohiban las leyes norteamericanas, tampoco serán elegibles para participar en la competencia.

Para comezar a participar los competidores que son elegibles deben tener una dirección postal válida, así como un correo electrónico válido para ingresar al registro. Todos los competidores deben registrarse y proveer toda la información requerida acerca de ellos en el sitio web de la competición, antes de participar en la misma.

1.3.4.3. Problemas

Durante cada ronda de la competencia, los competidores en la ronda serán presentados con un conjunto de problemas, de naturaleza algorítmica con una entrada para cada uno como se detalla más adelante. Una vez que una ronda comienza, cada competidor tendrá acceso a los problemas, y podrá descargar los archivos necesarios. Una solución válida para un problema consiste en lo siguiente, el código fuente del

competidor para solucionar a su criterio el problema, y la salida generada por el programa del competidor a partir de los datos de entrada proporcionados. Los envíos de las soluciones deben ser realizados en el tiempo especificado para cada problema, para que la solución sea considerada. Adicionalmente, los archivos de salida deben estar en el formato especificado por el sitio web de la competición o el reglamento.

Cuando un competidor intenta resolver un conjunto de entrada, un contador comienza tan pronto como el competidor descarga el archivo de entrada. El competidor tiene entonces 6 minutos para enviar la solución correspondiente, el archivo de salida junto con el código fuente utilizado para generarlo. El competidor puede enviar múltiples soluciones durante el período de 6 minutos, sin embargo, solo la última será considerada. Los resultados para los casos de prueba no serán revelados al competidor hasta que la ronda termine. Para cierto tipo de soluciones incorrectas, la solución será ignorada, el competidor será notificado con un mensaje, indicando que su solución no está bien formada. Los competidores no tendrán nuevas oportunidades para enviar nuevas soluciones luego de que el período de tiempo de 6 minutos haya concluido.

Los competidores deben incluir en su solución todo el código utilizado para generar la salida de cada problema. El tamaño máximo disponible para cada archivo es de 100KB. Los archivos fuentes pueden ser comprimidos pero su contenido luego de la descompresión no puede exceder 1MB en tamaño. El código deliberadamente ofuscado no estará permitido. Los competidores están advertidos de enviar sus soluciones con tiempo suficiente para ser recibido por los servidores de Facebook, para evitar problemas de latencia en la red que puedan afectar el envío de la solución. Los competidores no pueden solicitar reenviar soluciones, o reportar que enviaron una solución incorrecta luego de la ronda de competencia.

1.3.4.4. Puntuación

El sistema de puntuación se basa en los siguientes aspectos: cada problema vale la cantidad de puntos indicada en la parte superior del mismo. El puntaje total de

un competidor en una ronda, será la suma de los puntajes de todos los casos de prueba, para todos los casos de prueba que haya resuelto correctamente. En el caso de un empate entre dos competidores, dichos competidores serán ordenados en orden ascendente de acuerdo a los tiempos acumulados en sus soluciones. En otras palabras, en el caso de dos o más competidores con el mismo puntaje, el que tenga la menor cantidad de tiempo acumulado irá primero, y asi sucesivamente hasta el que tenga el mayor tiempo acumulado. El tiempo acumulado proviene de la suma del tiempo en el cual fue enviada la última solución enviada por un competidor, considerada correcta por los jurados para un problema (Dicho tiempo medido desde el inicio de la competencia).

Si luego de terminada una ronda, se alega alguna discrepancia entre el código fuente y la salida de algún competidor, en una solución considerada correcta, un panel de dos o más jueces, conformado por empleados de Facebook o sus subsidiarias, examinaran todos los códigos fuentes enviados por el competidor en esa ronda. Los jurados determinaran, en su absoluta discreción si existe una discrepancia, y de ser asi si es trivial o no. De resultar trivial al competidor se le anexaran 6 minutos en su tiempo para esa solución. En el caso de no ser trivial la discrepancia, el competidor pierde por forfeit todos los puntos acumulados en la ronda. En el caso de que los jurados no detecten ninguna discrepancia, el resultado permanecerá de la misma forma para el competidor.

Capítulo 2

Jueces Existentes

En este capítulo recaudamos información acerca de los jueces de maratones de programación existentes hasta la fecha. Cabe destacar que esta información es bastante escasa. Esto se debe a que la mayoría de estos jueces son proyectos personales, o proyectos académicos cerrados, donde la documentación acerca de su desarrollo es poca. La información aqui mostrada debe ser tomada como la base de una investigación acerca del ecosistema de jueces existentes actualmente.

2.1. PC ^ 2

Según la web de sus autores [18] fué desarrollado por la California State University, Sacramento (CSUS).

Está escrito en Java y está pensado para que funcione en cualquier plataforma Java 1.5, incluyendo Windows, Mac OS X y Unix.

Para permitir el desarrollo de competencias en diferentes sitios geográficos utiliza Java RMI (Java Remote Method Invocation) y sockets en su última versión.

Este juez está pensado para ser utilizado en competencia del tipo ACM-ICPC. Fué utilizado en estas competencias hasta el 2008, además de ser usado en las competencias regionales de este evento en 6 continentes.

2.2. Kattis

Como indican sus autores [19], fué desarrollado por KTH Royal Institute of Technology para corregir tareas de varios cursos de esta universidad. Luego sería usado en la final mundial de las competencias ACM-ICPC.

Kattis está escrito en una combinación de Python, PHP y SQL corriendo sobre Solaris.

Además de ser utilizado para corregir las tareas de variados cursos en la universidad KTH, actualmente es utilizado en la final mundial ACM-ICPC.

2.3. Hackzor

Es un proyecto alojado en Google Code [20] para realizar un juez de maratones de programación escrito en Python, utilizando el framework Django Web framework.

2.4. Boca

Según lo descrito en [21], Boca es un sistema de administración para realizar competencias de programación desarrollado en Brasil en la Universidad Católica de Sao Paulo y en la Universidad de Sao Paulo.

Está escrito en PHP y utiliza PostgreSQL como motor de bases de datos.

Soporta competencias de tipo ACM-ICPC y ha sido utilizado en un gran número de competencias Regionales de este evento.

2.5. Domjudge

Como aparece en [22], Domjudge es un proyecto que comenzó en el año 2004 en la Utrecht University, Holanda.

Fué escrito en lenguaje PHP y utiliza como base de datos MySQL

Soporta competencias de tipo ACM-ICPC, siendo uno de los jueces más utilizados para los maratones regionales de esta competencia.

2.6. Midas Judge

Según [23], Midas Judge es un sistema desarrollado en la Universidad de los Andes en Colombia para la realización de competencias de programación.

Escrito en C# utilizando .Net, es ofrecido como una alternativa de poca configuración para PC ^ 2.

El estándar que soporta es el de la competencia ACM-ICPC.

2.7. U WP Judging Tool

Según [24], es un proyecto de un sistema simple para la correción de problemas en los maratones de programación realizado en la Universidad de Wisconsin.

Consta de dos programas con interfaz gráfica de usuario, un servidor y un cliente, los cuales son utilizados para la interacción entre los equipos y el juez. No requiere instalación, aunque sus funcionalidades son un poco limitadas.

Está diseñado para la realización de competencias con el estandar ACM-ICPC

2.8. WACS

Según [25], es un proyecto de un juez de programación Web construido en la Khalifa University of Science, Technology and Research en los Emiratos Arabes.

Fue desarrollado utilizando PHP, CSS, Javascript, AJAX y MySQL.

Soporta competencias de tipo ACM-ICPC y fué pensado como un reemplazo para PC ^ 2.

Capítulo 3

Tecnologías Actuales

El sentido de este capítulo, es la recopilación de información acerca de las tecnologías utilizados para la realización de la plataforma web, capaz de cumplir con los requerimientos de un Juez para maratones de programación. Esta recopilación tiene como propósito, la exposición de los conceptos y arquitectura utilzados en la construcción de la pieza de software. Además de mencionar tecnologías, tambien se hace referencia a patrones, técnicas y otros conceptos básicos necesarios para la compresión de las necesidades de este tipo de aplicación.

3.1. Conceptos Básicos

- **Sandbox**

  Según [48], las compentencias de programación con evaluación automática de las soluciones enviadas, usualmente emplean un sandbox. Su trabajo es correr la solución en un ambiente controlado, mientras se hace cumplir la seguridad y se limitan los recursos.

  Muchas competencias de programación en el mundo implementan correción automática de los programas enviados por los competidores. Esto generalmente se logra corriendo las soluciones con lotes de datos de entrada y probando la correctitud de la salida. El programa además debe terminar cada caso de prueba

con ciertos límites de tiempo y memoria, esto para hacer posible la distinción entre soluciones correctas de diferente eficiencia.

Además de todo esto, se deben aplicar ciertas medidas de seguridad para evitar trampa por parte de los competidores, por ejemplo el programa no debe ser autorizado para accesar archivos para robar la respuesta correcta, terminar otros procesos ni comunicarse a través de la red.

Para lograr esta seguridad y limitar los recursos, los programas usualmente son corridos en un ambiente controlado llamado Sandbox.

- **API**

  Como se menciona en [26], una Interfaz de programación de aplicaciones es el conjunto de funciones y procedimientos (o métodos, en la programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción. Son usadas generalmente en las bibliotecas. Representa la capacidad de comunicación entre componentes de software. Un API especifica como algunos componentes de software deberían interactuar entre sí. En resumen un API es utilizado para definir y proveer acceso a otros componentes de software externos a una aplicación.

- **Autenticación basada en tokens**

  Según [57] el estándar en la web cuando se habla de autenticación en los últimos años es la autenticación básica basada en el servidor. Mientras la web, las aplicaciones y el auge de las aplicaciones móviles este tipo de autenticación ha mostrado problemas, especialmente en la escalabilidad.

  Debido a estos problemas, se cambió de enfoque y nació la autenticación basada en tokens.

  Este tipo de autenticación no tiene estado. No se almacena información acerca del usuario en el servidor o en una sesión.

En vez de almacenar los datos del usuario en una sesión, se genera y entrega al usuario un token de autenticación, el cual el usuario almacena y envía con cada cada solicitud para autenticarse.

- **JSON Web Tokens**

  Como aparece en [58], un JSON Web Token o JWT, es un estándar de autenticación basada en tokens, en la cual la información que contiene el token es transmitida utilizando JavaScript Object Notation o JSON.

  Funcionan en una gran variedad de lenguajes de programación como Python, Javascript, PHP, Ruby, Go, entre otros.

  Los JWT son auto-contenidos, es decir, contienen la información necesaria dentro de si mismos. Esto significa que un JWT es capaz de transmitir información básica de si mismo, una caga útil y una firma.

  Son fáciles de transmitir, dado que son auto-contenidos. Pueden usarse dentro de las cabeceras HTTP cuando se autentica con una API.

- **ORM (Object-Relational Mapping)**

  Según [27] el mapeo objeto-relacional es una técnica de programación para convertir datos entre el sistema de tipos utilizado en un lenguaje de programación orientado a objetos y el utilizado en una base de datos relacional, utilizando un motor de persistencia. En la práctica esto crea una base de datos orientada a objetos virtual, sobre la base de datos relacional. Esto posibilita el uso de las características propias de la orientación a objetos (básicamente herencia y polimorfismo). Esta técnica es utilizada para crear transparencia entre la capa de datos y la lógica de negocios a implementar en la aplicación, abstrayendo así al programador de la implementación utilizada para la manipulación de bajo nivel de los datos.

- **Base de datos orientada a objetos**

Citando a [28], en una base de datos orientada a objetos, la información se representa mediante objetos como los presentes en la programación orientada a objetos. Cuando se integra con un lenguaje de programación orientado a objetos, el resultado es un sistema gestor de base de datos orientada a objetos. Difiere de una base de datos relacional en que estas son orientadas a tablas. Estas bases de datos permiten al programador a desarrollar el producto, almacenarlo como un objeto y replicarlo o modificarlo para hacer nuevos objetos, facilitando así la manipulación de los datos para el programador.

- **Tecnología Push**

  Según [41] la tecnología push describe un estilo de comunicación basada en internet donde la solicitud para una transacción dada es iniciada por el servidor central. Es contrastada con la tecnología pull, donde la solicitud por la transmisión de información es iniciada por el cliente.

- **Comet**

  En [42] se describe Comet como un modelo de aplicación web en donde una solicitud HTTP sostenida durante mucho tiempo, permite a un servidor web hacer "push" de datos a un navegador, sin que el navegador la solicite explícitamente. Comet es un término utilizado para agrupar múltiples tecnologías utilizadas para lograr esta interacción.

- **WebSocket**

  Como se menciona en [43], un websocket es un protocolo que provee comunicación full-duplex sobre una simple conexión TCP. Está diseñado para ser implementado en navegadores y servidores web, pero puede ser usado por cualquier cliente o aplicación de servidor. Este protocolo forma parte de la definición del esta ndar HTML en su versión 5.

- **Aplicaciones de una página** Según [51], una aplicación de una página (Single Page Application en inglés) es una aplicación web o sitio web que encaja en una

sola página web con la meta de proveer una experiencia de usuario más fluida.

- **Aplicación Distribuida**

  Una aplicación con distintos componentes que se ejecutan en entornos separados, normalmente en diferentes plataformas conectadas a través de una red, es una aplicación distribuida. Las típicas aplicaciones distribuidas son de dos niveles (cliente-servidor), tres niveles (cliente-middleware-servidor) y multinivel. Una meta importante para los sistemas distribuidos es lograr la transparencia de localización. Un programa que se ejecuta en un ambiente distribuido se conoce como un programa distribuido, y la programación distribuida es el proceso de escribir dichos programas. Esto descrito según [29].

3.1.1. Arquitecturas de aplicaciones distribuidas

En la siguiente sección se describen las diferentes arquitecturas de aplicaciones distribuidas, según lo descrito en [29] y [30].

- **Arquitectura Cliente-Servidor** Es un modelo de aplicación distribuida en el que las tareas se reparten entre los proveedores de recursos o servicios, llamados servidores, y los demandantes, llamados clientes. Un cliente realiza peticiones a otro programa, el servidor, quien le da respuesta.

  La separación entre cliente y servidor es una separación de tipo lógico, donde el servidor no se ejecuta necesariamente sobre una sola máquina ni es necesariamente un sólo programa.

- **Tres niveles** Esta arquitectura es una extensión del modelo Cliente-Servidor que separa la lógica de negocios de los datos, creando una tercera capa.

- **N-Niveles** Se refiere típicamente a aplicaciones web que redirigen sus solicitudes a otros servicios empresariales. Este tipo de aplicación es la responsable del éxito de los servidores de aplicación.

- **Clusters** Se refiere a clusters de máquinas que trabajan juntas, corriendo un proceso compartido en paralelo y que pueden ser vistas como un solo sistema. Los componentes de un cluster generalmente son conectados entre sí a través de conexiones de área local (LAN) con cada nodo ejecutando su propia instancia de un sistema operativo.

- **Peer-to-peer** Es una arquitectura donde no hay máquinas especiales que proveen un servicio o manejan los recursos de la red. En cambio todas las responsabilidades son divididas uniformemente entre todas las máquinas, conocidas como peers.

- **Basada en el espacio** Se refiere a una arquitectura que crea la ilusión de un solo espacio físico.

- **Basada en servicios** Es un paradigma de arquitectura para diseñar y desarrollar sistemas distribuidos. Es un marco de trabajo conceptual que permite a las organizaciones unir los objetivos de negocio con la infraestructura de TI integrando los datos y la lógica de negocio de sus sistemas separados.

3.2. Patrones

Según [49], en la ingenieria de software un patrón de diseño es una solución general reusable para un problema recurrente en un contexto dado en el diseño de software. Un patrón de diseño no es un diseño finalizado que puede ser transformado directamente en código, sino una descripción o plantilla de como resolver un problema que puede ser usada en diferentes situaciones.

A continuación se describen los patrones utilizados según [31], [50]

- **MVC** es un patrón arquitectónico para el diseño de interfaces de usuario. Divide una pieza de software en tres partes interconectadas. El componente principal,

el modelo, consiste en los datos de la aplicación, reglas de negocio, lógica y funciones. La vista puede ser cualquier representación de la información que es mostrada al usuario. Y por último, el controlador acepta la entrada y la convierte en comandos para el modelo o vista.

- **Publicación-suscripción** es un patrón de mensajería donde los que envían mensajes, llamados editores o en inglés "publishers", no programan los mensajes para ser enviados directamente a receptores específicos. En cambio, los mensajes publicados son caracterizados en clases, sin conocimiento de los suscriptores que puedan haber. De forma similar, los suscriptores expresan interés en una o más clases, y solo reciben mensajes que son de su interés, sin conocimiento de los editores que puedan haber.

  Este patrón es utilizado en la pieza de software desarrollada por medio de la biblioteca Socket.IO 3.6 mencionada en la sección de bibliotecas.

Además de los patrones mencionados anteriormente, en este trabajo también se hace uso de otros patrones de diseño gracias al framework AngularJS 3.5, cuya descripción escapa del alcance de este trabajo.

3.3. Software utilizado

3.3.1. Lenguaje de Programación

Según [34], un lenguaje de programación es un lenguaje formal diseñado para expresar procesos que pueden ser llevados a cabo por máquinas como las computadoras.

- **JavaScript**

  Según [35], JavaScript es un lenguaje de programación interpretado. Se define como orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico.

Se utiliza principalmente en su forma del lado del cliente, implementado como parte de un navegador web permitiendo mejoras en la interfaz de usuario y páginas web dinámicas aunque existe una forma de JavaScript del lado del servidor [36].

3.3.2. Lenguaje de Marcado

Como se menciona en [37], utilizar un lenguaje de marcado es una forma de codificar un documento que, junto con el texto, incorpora etiquetas o marcas que contienen información adicional acerca de la estructura del texto o su presentación.

- **HTML**

  Como aparece en [38], el Lenguaje de Marcado Hipertextual hace referencia al lenguaje de marcado predominante para la elaboración de páginas web que se utiliza para describir y traducir la estructura y la información en forma de texto, así como para complementar el texto con objetos tales como imágenes. HTML 5 es la quinta revisión importante del lenguaje de la WWW, HTML.

- **CSS**

  Según [39], las hojas de estilo en cascada (Cascading Style Sheets) hacen referencia a un lenguaje de hojas de estilos usado para describir la presentación semántica de un documento escrito en lenguaje de marcas. Su aplicación mas común es dar estilo a páginas web escritas en lenguaje HTML y XHTML, pero tambíen puede ser aplicado a cualquier tipo de documentos XML.

3.4. Plataformas de Software

- **NodeJS**

  Según sus autores [44], NodeJS es una plataforma de software construida sobre el runtime"de Javascript de Chrome, para la construcción fácil de aplicaciones

escalables de redes. NodeJS usa un modelo manejado por eventos, con entrada y salida no bloqueante que lo hace de peso ligero y eficiente, perfecto para aplicaciones de datos intensivos y en tiempo real que corren a través de servicios distribuidos.

Debido a que el motor V8 de Google Chrome compila a código ensamblador, a diferencia de, por ejemplo, la plataforma Tornado de Python, NodeJS presenta una significamente mejora en tiempo de ejecución para rutinas de servidores, lo cual presenta una gran ventaja a la hora de trabajar en ambientes donde la concurrencia es vital para el funcionamiento ideal del sistema. Esto se ha comprobado a través de pruebas de rendimiento y recursos realizadas para comparar NodeJS y Python3 [45].

Otra de sus grandes ventajas es la existencia de un excelente manejador de paquetes, el cual viene incluído en la distribución de la plataforma, además de una comunidad bastante activa que permanentemente se encuentra publicando soluciones de software al repositorio de este manejador.

La existencia de librerías como Socket.IO viene a facilitar la tarea de crear aplicaciones web que utilicen tecnologías Comet, y gracias al manejador de paquetes de NodeJS, la inclusión de este tipo de depencias se convierte en tarea fácil tanto para el desarrollador como para el usuario que desea instalar la pieza de software que se contruya.

Para continuar nombrando ventajas, una de las más mencionadas es la capacidad de mantener solo un lenguaje de programación del lado del servidor y del lado del cliente, en este caso Javascript, lo cual hace que el desarrollo sea más sencillo ya que no es necesario el cambio de contexto a la hora de programar en los diferentes ambientes.

Como desventajas de esta plataforma tenemos que, por defecto, utiliza solo un hilo de procesamiento, lo cual desperdicia capacidad de procesamiento de muchos procesadores actuales, y para cambiar este comportamiento es necesaria la

instalación de otras herramientas y extensiva configuración.

- **Docker**

  De acuerdo a sus autores [53], es una plataforma abierta para desarrolladores y administradors de sistema para construir, enviar, y correr aplicaciones distribuidas. Consiste en el Motor Docker "Docker Engine", un motor portable, de tiempo de ejecución ligero y una herramienta de empaquetado, y "Docker Hub", un servicio en la nube para compartir aplicaciones y automatizar flujos de trabajo, Docker permite que las aplicaciones sean ensambladas rápidamente a partir de componentes y elimina la fricción entre los ambientes de desarrollo, pruebas y producción.

  Básicamente, Docker es una herramienta que puede empaquetar una aplicación y sus dependencias en un contenedor virtual que puede correr en cualquier servidor de Linux. Esto lo logra a través del kernel de Linux, libvirt, Linux Containers, cgroups y systemd-nspawn.

3.5. Frameworks

Según [40], un Framework define en términos generales, un conjunto estandarizado de conceptos, prácticas y criterios para enfocar un tipo de problemática particular que sirve como referencia, para enfrentar y resolver nuevos problemas de índole similar.

- **SailsJS**

  Es un framework de NodeJS que según sus autores [46] facilita la construcción de aplicaciones personalizadas, de grado empresarial utilizando NodeJS. Está diseñado para imitar el patron MVC de frameworks como Ruby on Rails, pero soportando los requerimientos de las aplicaciones modernas. Es especialmente bueno para construir chats, tableros en tiempo real o juegos multiplataforma.

Como sus funcionalidades principales tenemos: mapeo objeto relacional (ORM), generación automática de un API REST, provee seguridad básica y acceso basado en roles por defecto, además de la capacidad de agregar cuantas políticas de acceso sean necesarias; minificación automática de los archivos utilizados del lado del cliente. Todo esto además de las ventajas típicas de un framework moderno como son el soporte de sesiones, vistas genéricas y sistemas de plantillas.

Además de las funcionalidades anteriores, hay que destacar, que este framework viene integrado con la librería Socket.IO, la cual provee un API para la implementación de aplicaciones en tiempo real. Esto junto a que este framework utiliza NodeJS como base, nos provee de una solución del lado del servidor ideal para manejar multiples conexiones concurrentes y así poder construir una aplicaión de tiempo real sin mayor complicación. Esto se considera una gran ventaja sobre otros frameworks como Django, el cual requiere una gran configuración para lograr lo mismo.

Otra de las ventajas de este framework, es la incorporación de un servidor web propio optimizado para su rendimiento, lo que hace que no sea estrictamente necesaria la instalación de un servidor web como Apache o Nginx para su funcionamiento, aunque es recomendado para aplicaciones que reciban gran cantidad de carga. Esto lo aventaja sobre Django, ya que el servidor incorporado en Django, no es recomendado para ambientes de producción por su lentitud.

Para nombrar una de sus desventajas, SailsJS es un framework muy joven, por lo tanto, la comunidad desarrolladora en este framework es pequeña, así que es más complicado conseguir soporte de la comunidad a la hora de corregir errores o implementar funcionalidades ajenas al framework.

- **AngularJS**

  Según [52], es un framework para aplicaciones web de código abierto mantenido por Google y una comunidad de desarrolladores individuales y corporaciones para resolver muchos de los retos encontrados al desarrollar aplicaciones de una

sola página 3.1. Su meta es simplificar tanto el desarrollo como las pruebas a través de un marco de trabajo para el lado del cliente usando la arquitectura MVC 3.2.

Como funcionalidades principales, además de la implementación de un patrón MVC del lado del cliente, tenemos "Two way data binding", o unión de los datos en dos vías, lo que permite reflejar los cambios hechos en los datos automáticamente en la vista HTML lo cual facilita el desarrollo y hace la construcción de aplicaciones de una sola página bastante trivial. Además provee una estructura que permite la implementación de gran cantidad de patrones de diseño, incentivando así a la construcción de aplicaciones de calidad.

Incorpora un sistema de plantillas para el trabajo con las vistas, que permite la representación de los datos de una manera legible y sencilla.

Una de las desventajas más importantes señaladas por la mayoría de los autores en la web, además de su elevada curva de aprendizaje, son los problemas de rendimiento que muestra cuando se trabaja con grandes cantidades de data, aunque estos generalmente pueden ser evitados haciendo uso de patrones de diseño y las recomendaciones dadas por los desarrolladores de este framework.

3.6. Bibliotecas

- **Bootstrap**

  Es un framework de desarrollo web, que simplifica la forma de definir la estructura de una página web, a través de componentes escritos en CSS y en JavaScript, facilitando la construcción y el diseño de un sitio web.

  Define estilos e interacciones para una gran cantidad de elementos tradicionales web como botones, barras de navegación, enlaces, entre otros; para mantener un aspecto moderno. Además implementa un modelo de estilos adecuado tanto

para las webs de escritorio como móviles sin tener que duplicar el código base de la aplicación.

- **Socket.IO**

  Como aparece en [47], Socket.IO es una biblioteca de JavaScript para aplicaciones web de tiempo real. Tiene dos partes: una librería de lado del cliente que corre en un navegador, y una librería del lado del servidor para NodeJS. Ambos componentes tienen un API practicamente idéntico. Así como NodeJS, es manejado por eventos.

  Utiliza principalmente el protocolo WebSocket, pero si no está disponible puede utilizar otros métodos como los sockets de Adobe Flash, JSONP polling y AJAX long polling, mientras provee la misma interfaz. Aunque puede ser utilizado como una simple envoltura para WebSockets, provee muchas más funcionalidades, incluyendo radiodifusión a múltiples sockets, almacenamiento de datos asociados con cada cliente y una entrada y salida asíncrona.

  Esta biblioteca se encuentra incluida en el framework SailsJS como parte de su código para el manejo de eventos en tiempo real.

Capítulo 4

Planteamiento del problema

4.1. Justificación

A lo largo del estudio presentado podemos distinguir dos áreas en crecimiento, la primera es la variedad en las competencias de programación, desde las competencias presenciales, a las competencias online, cada una presenta sus particularidades en las reglas y evaluación.

Por otra parte, la intensa evolución en las tecnologías de desarrollo de software, frameworks y arquitecturas, hacen necesario evaluar, si muchas aplicaciones han cumplido su ciclo y deben ser reemplazadas, o si por el contrario, permanecen vigentes a pesar de los cambios.

En base a la investigación realizada con las aplicaciones existentes, se puede notar que muchas de ellas están diseñadas para un tipo específico de competencias, haciéndolas en su mayoría poco flexibles en este sentido. Adicionalmente, derivado de nuestra experiencia utilizándolas como competidores y como organizadores, se hace bastante compleja su instalación y configuración. Otro punto en contra, es que muchas de ellas fueron desarrolladas con tecnologías o técnicas hoy en día obsoletas.

4.2. Objetivo General

Por lo anteriormente expuesto, hemos decidido realizar, el diseño e implementación de una plataforma que tenga por objetivo proveer los requerimientos necesarios para realizar una competencia de programación.

4.3. Objetivos Específicos

- Analizar los sistemas existentes en la actualidad, verificando los requerimientos necesarios para un juez de competencias de programación.
- Realizar el diseño del sistema considerando características de seguridad, facilidad de uso e instalación, usabilidad y coordinación distribuida presentes en otros sistemas ya existentes.
- Implementar un sistema que permita gestionar las competencias de programación con una arquitectura distribuída y comunicación vía sockets entre todos los actores de la competencia.
- Garantizar seguridad y tolerancia a fallos en todos los momentos de la competencia.
- Realizar pruebas de rendimiento y de aceptacion para garantizar la calidad del sistema

4.4. Arquitectura

En aras de cumplir los objetivos mencionados, planteamos una arquitectura orientada a servicios para el proyecto, permitiendo el trabajo de forma distribuída. Utilizando tecnologías modernas, podemos introducir mejoras en la comunicación entre los

competidores y los jurados, utilizando websockets, tecnología ampliamente utilizada en la actualidad.

Para la organización del desarrollo, utilizaremos el patrón de diseño Modelo Vista Controlador, con el fin de separar lógicamente los componentes que integrarán la aplicación, permitiendo una mayor agilidad en el desarrollo.

En cuanto a los lenguajes de implementación, planteamos la utilización de Javascript como lenguaje principal, utilizando el motor de NodeJS, y como framework de trabajo SailsJS, todo ello, debido a la gran aceptación que están teniendo en la actualidad y la flexibilidad a la hora de realizar el desarrollo. Adicionalmente para agregar valor del lado del cliente de la aplicación utilizaremos el api de Jquery, y para el diseño de las vistas utilizaremos la librería Bootstrap, con el fin de lograr una gran interfaz de usuario para la aplicación.

4.5. Metodología

Como metodología de trabajo, aplicaremos el uso de las metodologías ágiles, en particular Scrum y XP, con la finalidad de lograr iteraciones rápidas, que permitan ir viendo de forma continua el desarrollo, permitiendo una mayor precisión en cuanto al mismo, basado en las reacciones de los usuarios.

Capítulo 5

Implementación

Este capítulo tiene como objetivo exponer el proceso de la implementación de la pieza de software descrita como objetivo de este trabajo. Para mantener un nivel de organización apropiado, hemos dividido este segmento en seis secciones. Al ser una aplicación distribuida, se hace necesario comenzar desglosando su arquitectura y cada una de las partes que la componen. Luego para continuar, se describe el modelo de datos construido para la aplicación. En la siguiente sección se expone el proceso de sincronización entre los componentes de la aplicación, el cual viene a jugar un papel protagónico en el funcionamiento de la aplicación. Para continuar se describen los procesos mas importantes que forman parte del ciclo de vida de la aplicación. Para terminar se explica como el código del lado del cliente se comunica con el servidor para mostrar la información necesaria para los usuarios y el esquema de seguridad utilizado en dicha conexión.

El código fuente de la aplicación se encuentra publicado en un repositorio Git bajo la modalidad de código abierto. La dirección de dicho repositorio es `https://github.com/moosejs/moosejs`.

5.1. Arquitectura

En esta sección se describen los principales componentes de la aplicación y sus roles, para luego pasar a la revisión de su funcionamiento y ciclo de vida.

Se optó por basarnos en la arquitectura para aplicaciones distribuidas Cliente-Servidor, repartiendo así la carga de las diferentes tareas del sistema entre los diferentes proveedores de servicios.

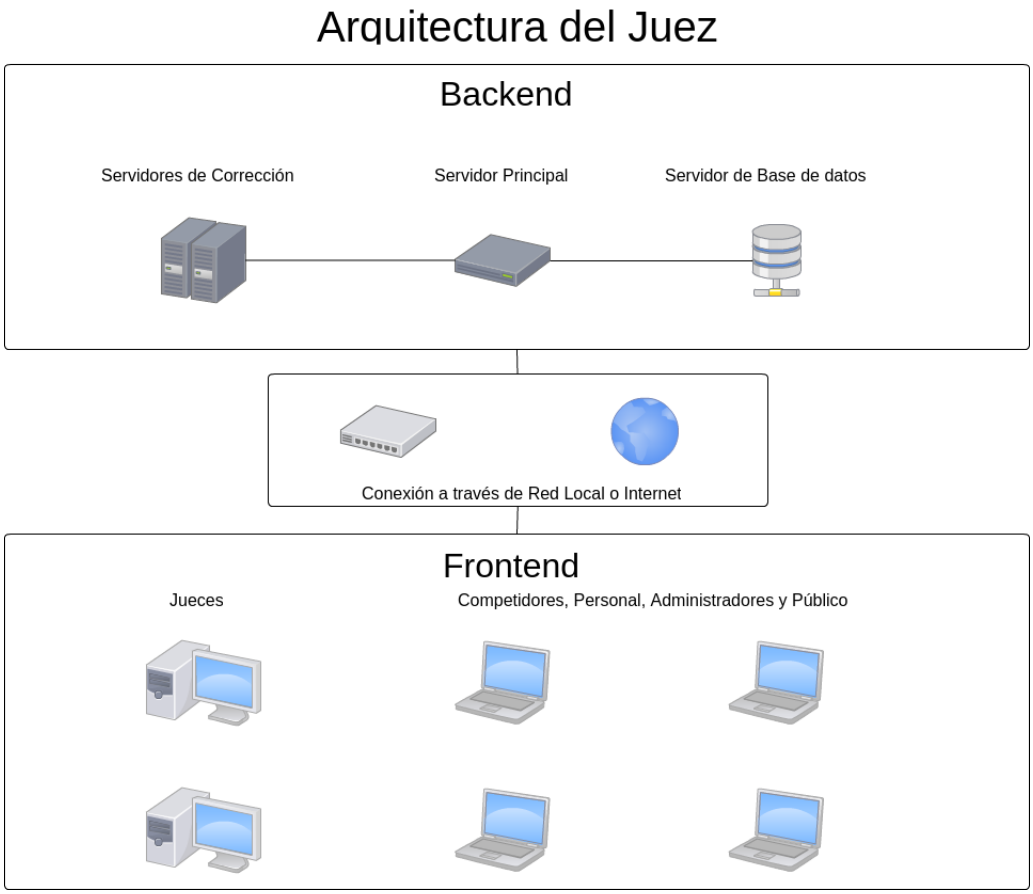

Figura 5.1: Diagrama de la arquitectura del juez

El backend del sistema está dividido en tres componentes principales:

- **Servidor Principal**

  Este servidor viene a ser la columna vertebral del sistema y su punto de entrada principal. Consiste en un servidor web escrito en el lenguaje de programación Javascript 3.3.1 y que corre sobre la plataforma NodeJS 3.4 utilizando el framework SailsJS 3.5. Entre sus responsabilidades están recibir las peticiones de

los clientes y darles respuesta; manejar, sincronizar y distribuir la carga entre los servidores de corrección y obtener la información almacenada en la base de datos.

- **Servidores de Corrección**

  Estos servidores son los encargados de la corrección de las soluciones enviadas por los competidores del evento. Para hacer esto, hacen uso de un script escrito en el lenguaje de programación Javascript 3.3.1 y que corre sobre la plataforma NodeJS 3.4. Este script se sincroniza con el servidor principal para obtener toda la información necesaria para la corrección de los problemas y espera por tareas de corrección. Pueden haber varias instancias de este script corriendo en la misma computadora o en computadoras diferentes.

- **Servidor de Base de datos**

  Este servidor se encarga del manejar los datos necesarios para el funcionamiento de la aplicación. Gracias al ORM 3.1 implementado en el framework SailsJS 3.5, el sistema no está atado a un sistema manejador de base de datos en específico, por lo tanto virtualmente cualquier manejador de base de datos puede constituir este componente.

Cabe destacar que los componentes aquí descritos representan partes lógicas de la aplicación, más no físicas. Por lo tanto estos componentes podrían convivir en la misma computadora servidor, o en diferentes computadoras. Esto con la intención de permitir escalabilidad en el sistema, en forma de escalabilidad horizontal al trasladar los servidores de correción o de base de datos de la computadora principal a computadoras adicionales, aligerando así la carga del servidor principal.

Como se muestra en la figura 5.1, la conexión entre el backend y el frontend del sistema, se puede realizar a través de una Red Local o Internet, dependiendo de la naturaleza de la competencia que se quiera realizar. Sin embargo, recomendamos

que la conexión entre los componentes del backend sea **siempre** a través de una Red Local ya que el flujo de datos entre estos componentes es significativamente alto y es importante contar con una red segura y consistente.

5.2. Modelo de datos

A continuación se expone el modelo de datos utilizado para representar los diferentes objetos o entidades que componen la información manejada por el sistema en su totalidad.

Elegimos un diagrama de Entidad Relación, a pesar de que el sistema no está ligado a un manejador de base de datos en particular, ya que es la forma mas sencilla de representar gráficamente este modelo.

Con el propósito de mejorar la legibilidad, el diagrama se muestra dividido en "módulos"que representan las funcionalidades de la aplicación.

5.2.1. Funcionalidades principales

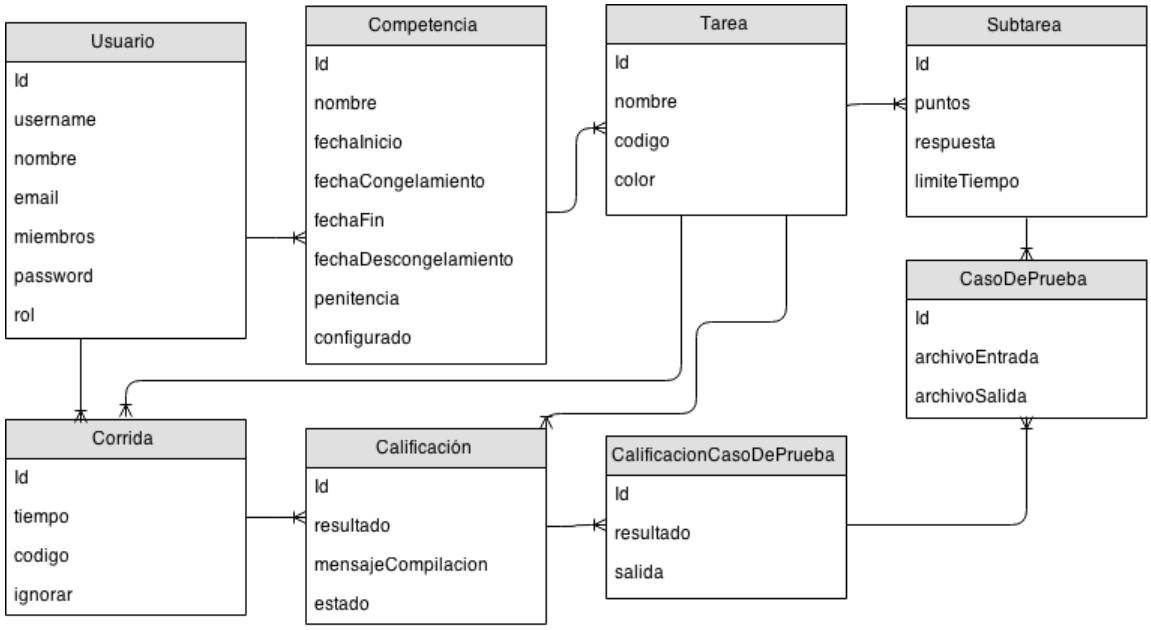

Figura 5.2: Diagrama Entidad Relacion Principal

Para la funcionalidad principal del sistema, que en este caso es el envío y corrección de una solución, se hacen necesarias las entidades observadas en la figura 5.2.1. En el diagrama anterior es importante hacer incapié en las siguientes observaciones:

- Para permitir soporte a multiples competencias ocurriendo simultáneamente, se crea la relación Usuario - Competencia (1 - N).
- Las entidades *Subtarea* y *CasoDePrueba*, proveen la estructura necesaria para soportar estándares con múltiples subtareas como el estándar IOI 1.2 o Google Code Jam 1.3.3 por nombrar algunos.
- El atributo *penitencia* de la entidad *Competencia* representa la cantidad de tiempo que se ha de agregar al tiempo final del equipo por cada envío incorrecto.

- El atributo *puntos* de la entidad *Subtarea*, permite definir la cantidad de puntos que el equipo debe obtener al resolver una subtarea en particular.

- El atributo *respuesta* de la entidad *Subtarea*, indica si la calificación de esa subtarea debe enviarse como respuesta al equipo y ser mostrada en el marcador, o si debe ocultarse hasta el final de la competencia.

5.2.2. Marcador

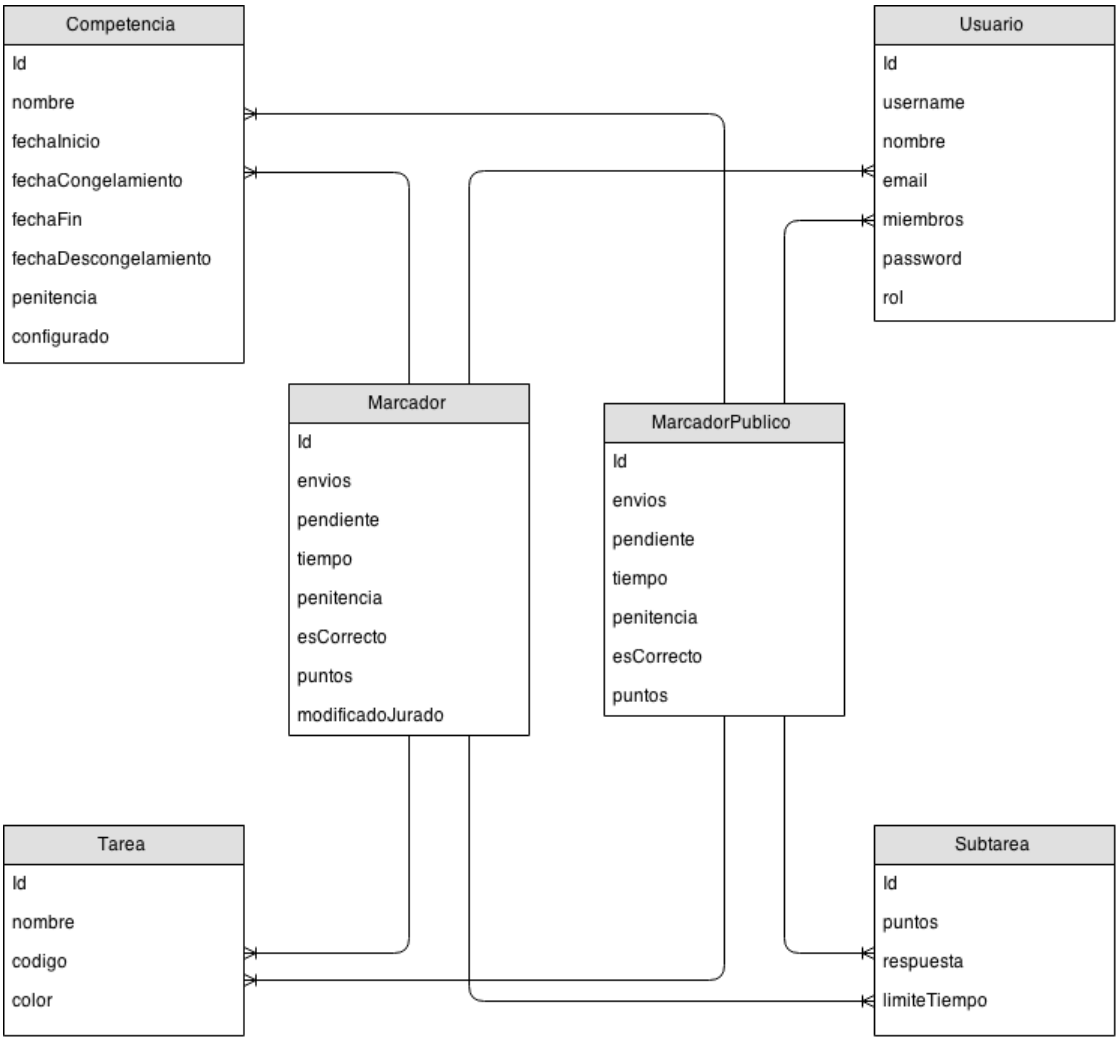

Figura 5.3: Diagrama Entidad Relacion Marcador

Para el funcionamiento del marcador, se utilizan dos entidades dedicadas a esta funcionalidad. Esto con la intención de agilizar la búsqueda y despliegue del marcador. Estas entidades tendrían una función parecida a una memoria caché, por lo que su existencia implica redundancia de datos.

Se utilizan dos entidades, una para el marcador privado, observado por el grupo de jurados de la competencia, y una para el marcador público, que sería observado por el resto de los usuarios (competidores, administradores, staff y público en general). La única diferencia entre ambas, además de la naturaleza de la información que mantienen, es el atributo *modificadoJurado* de la entidad *Marcador* que indica si la respuesta obtenida por el sistema fué modificada por alguno de los jurados.

5.2.3. Verificación de Soluciones

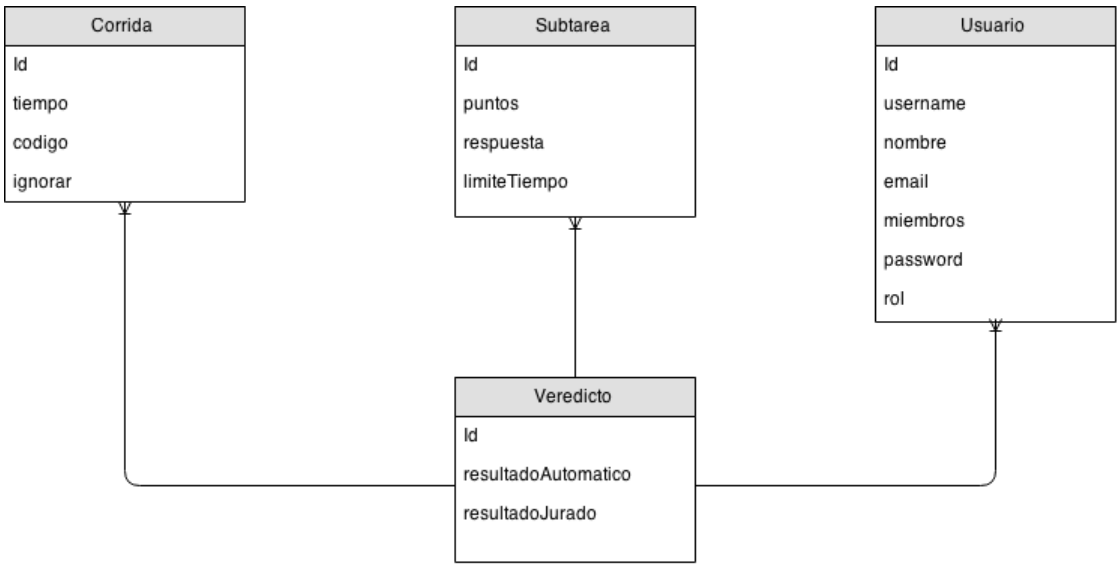

Figura 5.4: Diagrama Entidad Relacion Verificación de Soluciones

Para la verificación de soluciones por parte de los jurados, se introduce una entidad llamada *Veredicto* que contiene la respuesta final seleccionada por el jurado y la

respuesta obtenida por el sistema. Los *Veredictos* son creados por cada subtarea para permitir dar respuesta personalizada por cada subtarea.

5.2.4. Servidores de Corrección

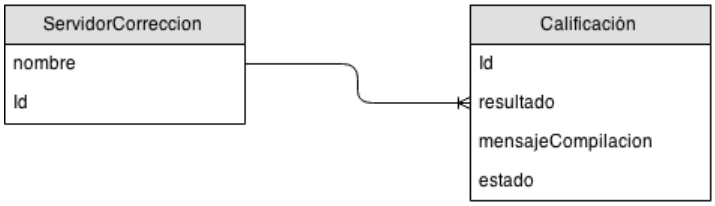

Figura 5.5: Diagrama Entidad Relacion Servidores de Corrección

Con el propósito de manejar los servidores de corrección, se utiliza la entidad *ServidorCorreccion*. Esta entidad tiene la utilidad de permitir la conexión y autenticación de los servidores, además de almacenar las *Calficaciones* corregidas por cada servidor de corrección.

5.2.5. Lenguajes de Programación

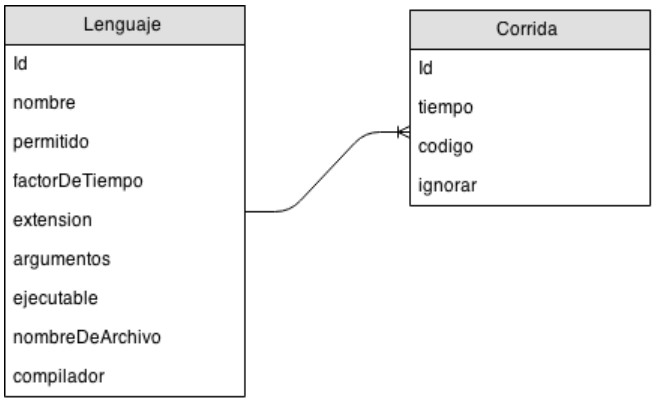

Figura 5.6: Diagrama Entidad Relacion Lenguajes de Programación

Para permitir la utilización de diferentes lenguajes de programación, se crea la entidad *Lenguaje*, que define el lenguaje de programación utilizado para realizar cierta *Corrida*

5.2.6. Aclaraciones

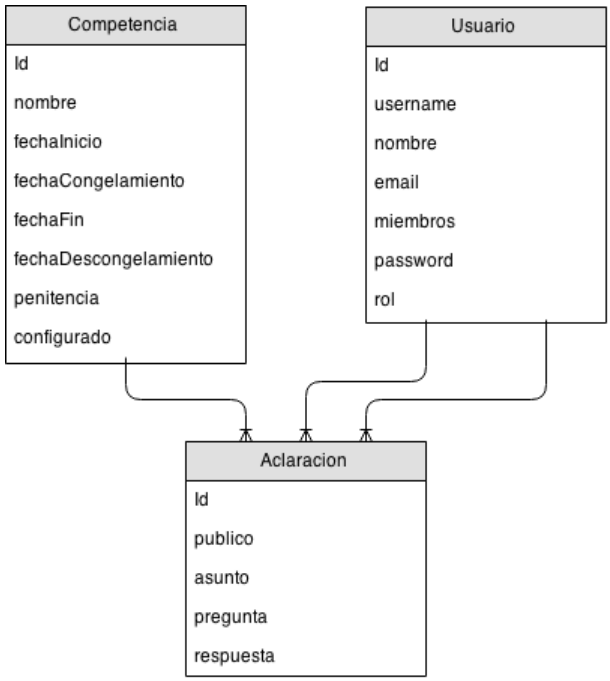

Figura 5.7: Diagrama Entidad Relacion Aclaraciones

En algunos estándares de competencias de programación se permite a los competidores hacer preguntas o solicitar aclaraciones, que deben ser contestadas por los miembros del jurado. Para incluir esta funcionalidad, se introduce la entidad *Aclaración* que almacena la pregunta realizada y su respuesta en caso de haber una. También mantiene el asunto o motivo de la solicitud de aclaración y si dicha aclaración debe ser vista por todos los participantes o solo el que la solicitó a través del atributo *publico*. En la figura 5.2.6 se observa que la entidad *Aclaración* posee dos relaciones con

la entidad *Usuario*, esto se hace con la intención de almacenar tanto el usuario que realiza la solicitud de aclaración, como el que la responde.

5.2.7. Resto de Funcionalidades o Funcionalidades Menores

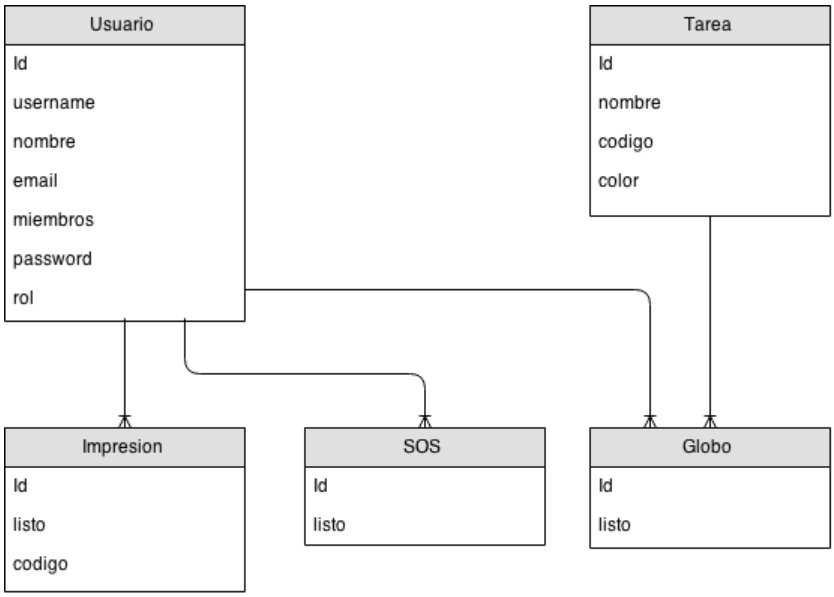

Figura 5.8: Diagrama Entidad Relacion Funcionalidades Menores

En el diagrama presentado en la figura 5.2.7 se ilustra el modelo de datos para el resto de las funcionalidades del sistema, como lo son:

- Notificaciones de entrega de globos para el estándar ACM 1.1 y similares.
- Llamada de auxilio al personal.
- Impresiones de código o material por parte de los competidores.

5.3. Roles

En gran parte de los sistemas que poseen un módulo de registro y acceso de usuarios, se hace necesario mantener un sistema de acceso en base a roles, para limitar los privilegios que tiene un usuario en particular en un sistema.

En el caso de nuestro sistema, estos roles pasan a ser similares a los roles que se ven en cualquier competencia de programación. Los roles implementados en el sistema desarrollado son los siguientes:

- **Competidor o equipo**

  Este es el rol más natural dentro de una competencia de programación. Se le permite enviar soluciones a los problemas, hacer solicitudes de aclaración y ver el marcador. Se le restringe el acceso a los módulos de configuración y manejo de la competencia.

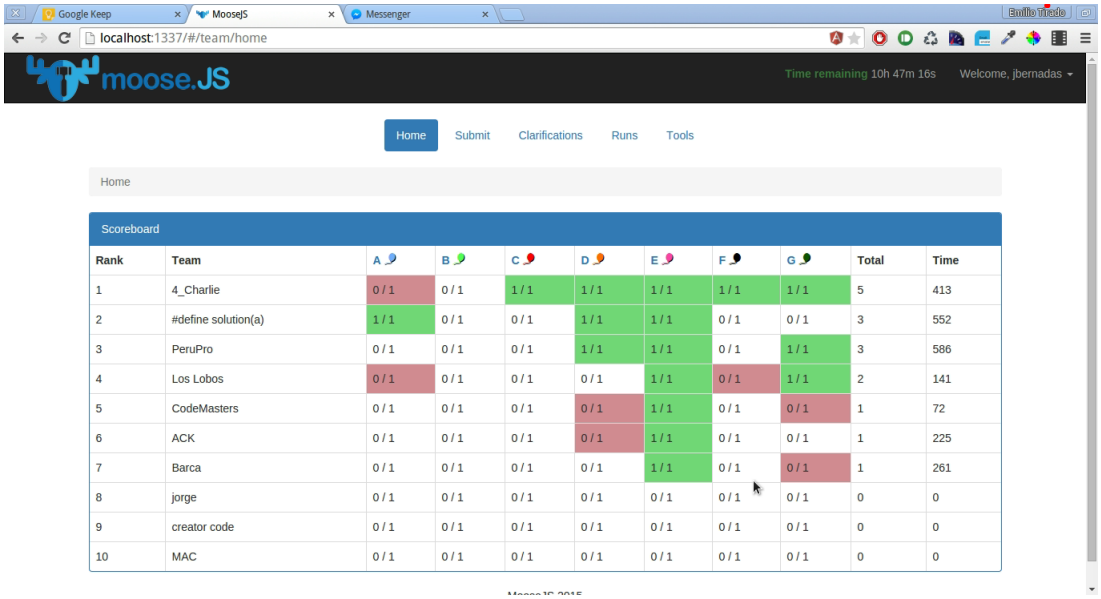

Figura 5.9: Vista principal del competidor

- **Administrador**

Es la persona encargada de las tareas de más bajo nivel, como instalar el sistema, los servidores de corrección, inscribir a los usuarios y crear las competencias.

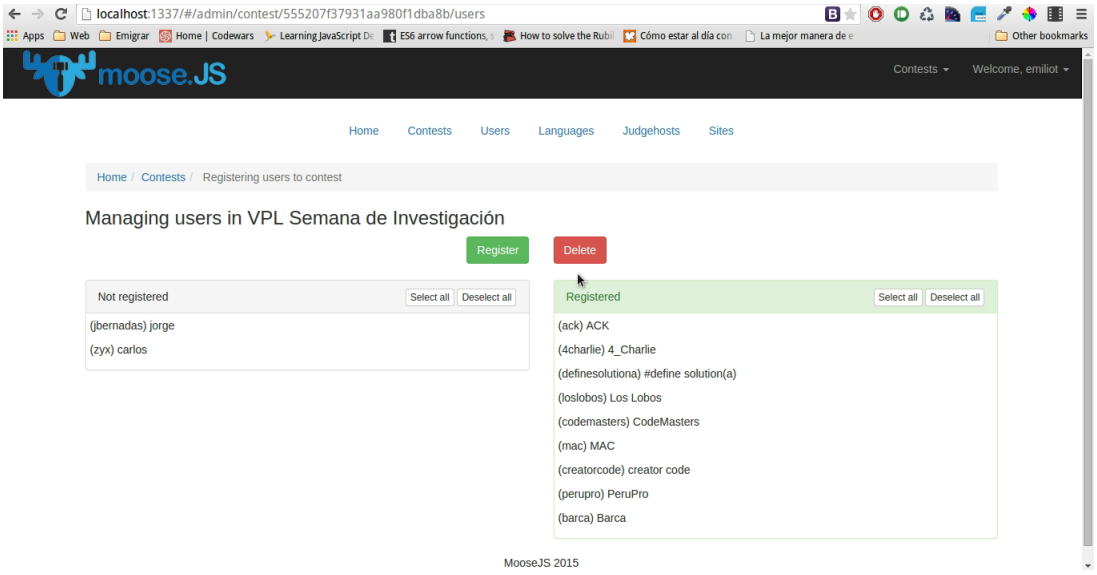

Figura 5.10: Vista de usuarios del administrador

- **Jurado**

  A este grupo de usuario se les atribuye la configuración de la competencia y todo lo relacionado con el desarrollo de la misma. Entre estas tareas están la escritura de los problemas y la configuración de los mismos dentro del sistema, responder a las solicitudes de aclaración y dar respuesta final a las soluciones.

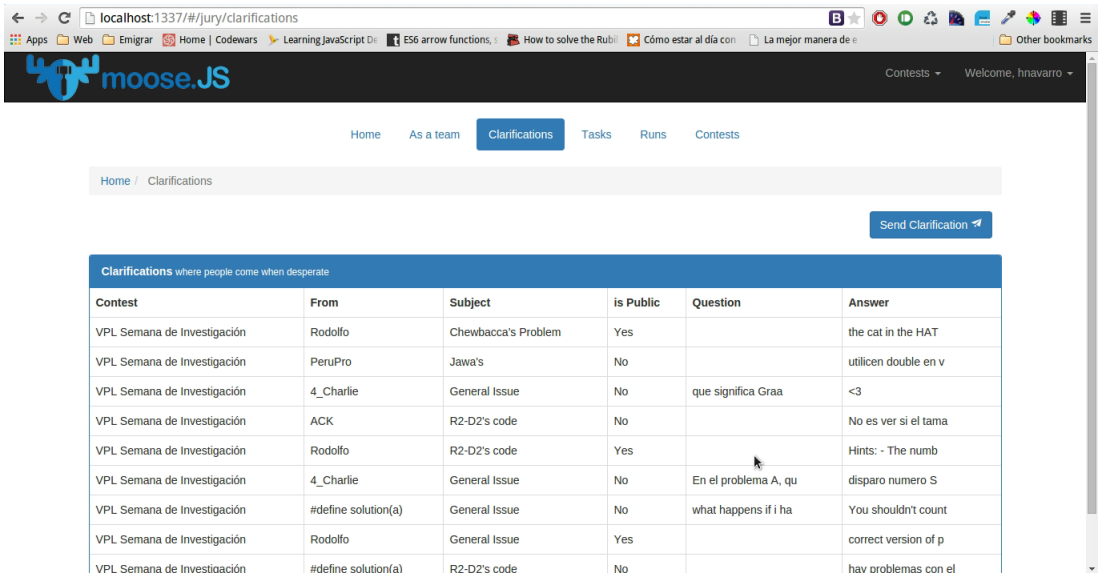

Figura 5.11: Vista de las solicitudes de aclaración por los jurados

- **Personal**

  El personal de apoyo dentro del sistema tiene como responsabilidad atender las solicitudes de ayuda, impresiones y el manejo de los globos.

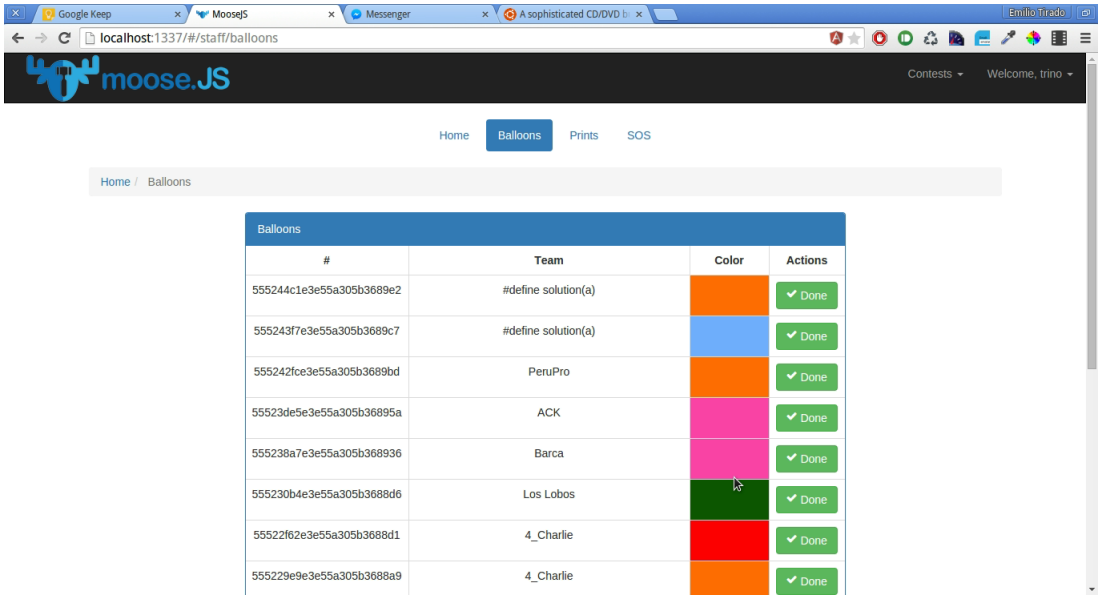

Figura 5.12: Vista de los globos desde el personal de apoyo

Es importante que todos los usuarios están en capacidad de revisar el marcador, así como también los usuarios que no se encuentran registrados y que se encuentran en la vista pública. Para todos los usuarios, el marcador que se muestra es el mismo, excepto para el jurado, cuyo marcador se muestra con los resultados finales de la competencia.

5.4. Sincronización

Cuando se trabaja en ambientes distribuidos, es necesario asegurar que la información requerida para el desarrollo de los procesos principales de los componentes de la aplicación distribuida sea confiable. Para esto se utilizan diferentes esquemas de sincronización de la información para asegurar esto en todo momento.

En esta sección presentamos el esquema de sincronización utilizado tanto para mantener sincronizados los servidores de corrección, como el frontend y las vistas que visualizan los usuarios.

Para poder explicar como se maneja la sincronización en cada uno de los componentes, es necesario primero explicar cuales son los datos que se deben sincronizar, en que formato y donde están almacenados.

Es importante destacar que muchas de las interacciones e intercambios de información se realizan a través de la biblioteca Socket.IO 3.6. Esto para simplificar el proceso de sincronización y delegar el trabajo de bajo nivel a esta biblioteca que ha demostrado ser confiable.

5.4.1. Servidores de Corrección

Los datos que necesitan ser sincronizados en este componente en particular, son los inherentes a la corrección de los problemas, en otras palabras, los archivos de entrada y salida de cada caso de prueba almacenado al sistema.

Cada servidor de corrección se encuentra ejecutando su propia copia del script mencionado en la sección Arquitectura 5.1. Dicho script esta compuesto por dos módulos, el módulo principal que se encarga de la sincronización y la comunicación de los datos con el servidor principal, y otro módulo que se encarga de compilar (si es necesario) y ejecutar las soluciones que será cubierto en la próxima sección de este capítulo.

El módulo principal de este script que llamaremos "daemon.º demonio, maneja la sincronización de la siguiente manera:

- Guarda en un archivo de configuración temporal, el momento en que se realizó la última actualización.
- Al lograr establecer una conexión con el servidor principal, ejecuta una función llamada *syncTestcases* que se encarga de realizar una petición al servidor con la fecha de la última actualización. La respuesta a esta petición es la información de todos los casos de prueba modificados o creados desde esa fecha de haber alguno. Esta información se encuentra almacenada en la base de datos del sistema. Con

esta información el daemon luego procede a obtener los archivos de entrada y salida de cada caso de prueba obtenido en el paso anterior, esto a través de una función llamada *getTestCase*.

- Luego del proceso descrito anteriormente, se puede considerar que los datos se encuentran en sincronía. En caso de que se modifique alguno de los datos sincronizados anteriormente, el daemon mantiene una conexión abierta con el servidor a través de Websockets 3.1 utilizando Socket.IO 3.6, por la cual se transmitiran los cambios en caso de que ocurran algunos mientras el daemon se está ejecutando.

- El proceso mediante el cual el daemon recibe peticiones de corrección será explicado en la siguiente sección de este capítulo.

5.4.2. Frontend

En el caso del Frontend, se mantiene sincronización en los datos que requieren atención oportuna por parte de los usuarios. Un ejemplo de esto, es la sincronización de las soluciones enviadas por los participantes en la vista de los jurados. Para lograr este tipo de comunicación en tiempo real, se utiliza la biblioteca Socket.IO 3.6, más específicamente la incluída en el framework SailsJS 3.5, que no es más que una iteración de más alto nivel del API original de Socket.IO.

En la mayor parte de la aplicación se utiliza este esquema de sincronización para actualizar los cambios ocurridos en objetos y entidades que pertenecen al usuario, o que por alguna razón dichos cambios le conciernen. Los cambios y objetos mencionados se describen a continuación en forma de eventos:

- Envío de soluciones por parte de los usuarios en la interfaz del jurado.

- Actualización del resultado de una solución en la interfaz del jurado y del competidor.

- Creación y modificación de solicitudes de aclaraciones tanto en la interfaz del jurado como del competidor.
- Actualización del marcador en la interfaz de cualquier usuario identificado del sistema, o de acceso público.
- Creación y culminación de tareas para el personal (globos, impresiones o llamadas de auxilio).

Existen dentro de la aplicación dos funcionalidades que cambian un poco este esquema. Estas son:

- Contador de eventos mostrado en todas las interfaces de los usuarios que se encuentran identificados en el sistema. Esto permite mostrar un contador de nuevos eventos al lado del nombre de cada sección, en las pildoras de navegación de la interfaz. Para hacer esto, el código del lado del cliente se suscribe a los eventos de creación, modificación y eliminación de los objectos a los cuales el usuario debería ser capaz de acceder.
- Temporizador de las competencias. Esta funcionalidad se puede apreciar en la barra de navegación principal de la interfaz. Muestra el tiempo restante para el próximo evento dentro de la competencia. Estos eventos son: comienzo, congelamiento del marcador, culminación de la competencia y entrega de resultados finales. Para esto se subscribe al cliente a los eventos de creación y modificación de las competencias, para estar en sincronía con los tiempos de la misma.

5.5. Procesos

Esta seccion tiene como propósito ilustrar los principales procesos que forman parte del ciclo de vida de la aplicación. Aquí diseccionaremos los procesos complejos que forman parte del desarrollo de un maraton de programación de cualquier estándar.

5.5.1. Crear y configurar una competencia

Esta es una de las tareas cotidianas dentro de un sistema de administración de competencias. Para comenzar un administrador crea la competencia asignándole un nombre. Luego, cualquier miembro del jurado puede entonces proceder a la configuración de la competencia. Configurar una competencia implica: asignar nombre, tiempo de inicio, hora muerta y final de la competencia; tiempo de penalización por malas entregas y la cantidad de problemas que tendrá la competencia.

Además de todo esto, el jurado deberá indicar la información de configuración de los problemas. Esto incluye el nombre y código de cada problema, el color del globo que se debe entregar en caso de completar el problema y la información de cada una de sus subtareas. La información de una subtarea incluye la cantidad de casos de prueba, si el resultado debe mostrarse públicamente y el tiempo límite para la ejecución de una solución, además de los archivos de entrada y salida de referencia de cada caso de prueba.

Para facilitar todo este proceso, se desarrollaron dos métodos para ingresar todos estos datos al sistema. El más directo consiste en la utilización de un archivo comprimido con una estructura en particular explicada en la interfaz de la aplicación, en el cual se encuentran los archivos necesarios para cada problema y un archivo principal de configuración, que utiliza la notación de objetos de Javascript.

El otro método para ingresar estos datos, es a través de un ayudante que simplifica la construcción de la estructura de la competencia, estó utilizando un formulario de múltiples pasos para no sobrecargar la interfaz de campos de texto. La desventaja de este método, es que luego es necesario subir los archivos de casos de prueba y los enunciados individualmente a través de la interfaz.

5.5.2. Subproceso de Compilación

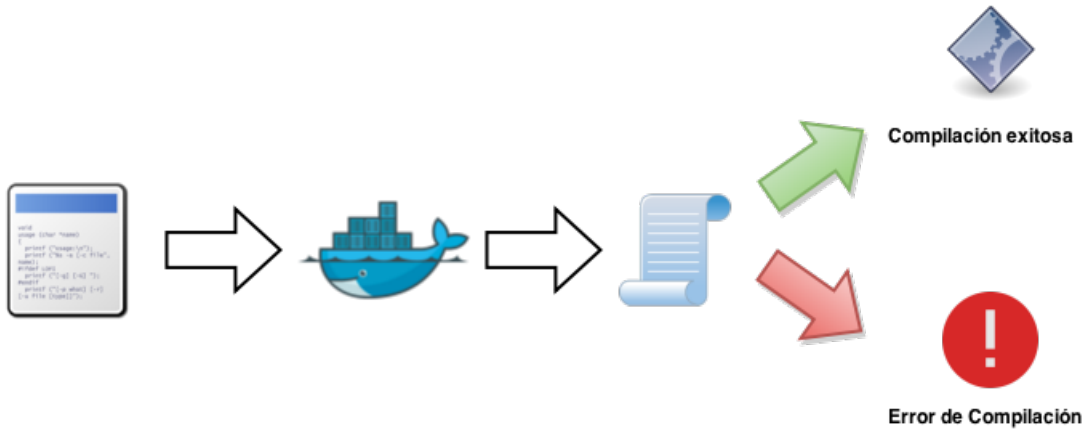

Figura 5.13: Diagrama Subproceso de Compilación

Este subproceso es ejecutado por un módulo dentro del demonio y se describe a continuación:

1. Se copia el código fuente al área de trabajo de este módulo.
2. Se copian unos scripts escritos en bash que se encargan de la compilación y la ejecución.
3. Se crean unos directorios que contendran las salidas y los posibles errores obtenidos de la compilación y ejecución.
4. Si el lenguaje utilizado por el competidor para desarrollar la solución es interpretado, el subproceso culmina en este paso.
5. Se crea un contenedor de Docker con la imagen desarrollada y se le ordena ejecutar el script encargado de compilar el código fuente.
6. Se retorna el resultado de la compilación al proceso invocador y se termina el subproceso.

5.5.3. Subproceso de Ejecución

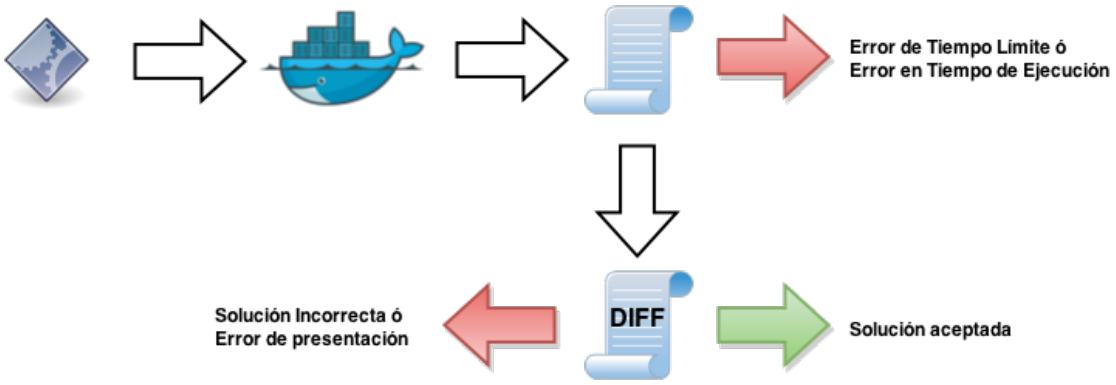

Figura 5.14: Diagrama Subproceso de Ejecución

Debido a que el Subproceso de Compilación 5.5.2 se asegura de que los prerequisitos para la ejecución de este subproceso, estén satisfechos, este subproceso es bastante trivial. Es ejecutado por el mismo módulo que el Subproceso de Compilación 5.5.2 y se describe a continuación:

1. Se crea un contenedor de Docker con la imagen desarrollada y se le ordena ejecutar el script encargado de ejecutar el código fuente

   Nota: para asegurar que la ejecución del código no exceda el tiempo límite impuesto por la subtarea, se utiliza un script llamado *TimeScript* el cual se encarga de limitar el tiempo de ejecución de un proceso a un máximo de milisegundos.

2. Luego de haber obtenido una respuesta de dicho script, se procede como sigue:

   (a) Si el código excedió el tiempo límite de ejecución impuesto por la subtarea, se retorna un error de tiempo excedido.

   (b) En caso de haber detectado algún otro error, se retorna un error de tiempo de ejecución

(c) Si no se detectó ningún error, se utiliza la herramienta *diff* para validar que la salida producida por la solución es la correcta. En caso de que dicha salida no coincida con la salida de referencia indicada por el jurado, se ejecuta de nuevo la herramienta con diferentes opciones para determinar si es un error de presentación o la respuesta es incorrecta.

(d) Luego de haber determinado la naturaleza de la solución, se retorna dicha respuesta y se culmina el subproceso.

5.5.4. Corrección de una solución

Este proceso ocurre dentro de los servidores de corrección, y está plasmado en el script que hemos llamado "daemon.º demonio. Este proceso puede ser invocado de tres maneras:

- El demonio se encuentra ocioso y llega un mensaje por medio de los Websockets 3.1 indicando que hay una nueva solución esperando corrección, luego se dispara este proceso.
- El demonio se encuentra desconectado por que el script no se encuentra corriendo o la conexión entre el servidor principal y el de corrección falló. Cuando el demonio logra establecer la conexión (asumiendo que los procesos de sincronización no son necesarios o ya se ejecutaron), el demonio se asegura de que no exista una solución esperando por corrección y en caso de que exista, se ejecuta este proceso.
- El demonio termina de corregir una solución anterior, luego se asegura de que no exista una solución esperando por corrección y en caso de que exista, se ejecuta este proceso.

Para soportar este proceso en su totalidad, se desarrolló una imagen de Docker 3.4 basada en el sistema operativo Ubuntu, en la cual se encuentran los compiladores y las

herramientas necesarias para la compilación y ejecución de las soluciones enviadas por los participantes. Docker 3.4 permite la creación de contenedores capaces de ejecutar tareas en un ambiente virtualizado. Gracias a esto, dichas tareas solo conocen su existencia y no tienen la posibilidad de interactuar con el proceso padre. Por lo anterior esta solución se hace más que adecuada para los procesos de Compilación y Ejecución, ya que provee la seguridad necesaria para dichos procesos sin mucha complicación.

Luego de que este proceso es invocado, ocurre lo siguiente:

Figura 5.15: Diagrama corrección de una solución

1. La corrida es marcada como "en proceso"
2. Se obtiene el código fuente de la solución a través de una solicitud Http
3. Se invoca al subproceso de Compilacion.
   (a) Si la compilación no fue exitosa, se guarda la salida del compilador y se termina el proceso.
   (b) En caso de que la compilación haya sido exitosa, se invoca a la función *judgeTask*.

4. La función *judgeTask* se encarga de iterar las subtareas de la tarea asociada con la corrida para llamar a la función *judgeSubtask* con cada una de ellas.

5. De forma análoga a la funcion *judgeTask*, la función *judgeSubtask*, itera los casos de prueba de cada subtarea para invocar a la función *judgeTestcase* con cada uno de ellos.

6. Dicha función invoca al subproceso de Ejecución con los datos específicos provistos por la subtarea y el caso de prueba (limite de tiempo, archivos de entrada y salida).

7. Luego de culminar dicho subproceso, se comunica al servidor principal el resultado obtenido.

8. Al terminar de iterar cada subtarea con sus respectivos casos de prueba, la función *judgeTask*, marca la corrida como lista y culmina el proceso.

5.5.5. Ciclo de vida de una solución

Este proceso es el más importante de los expuestos aquí, ya que en otras palabras, son todos los pasos que ocurren desde que un competidor decide enviar una solución, hasta que el jurado da respuesta a la misma.

Para mantener la lista de servidores de corrección conectados, se utilizan canales de comunicación provistos por la biblioteca Socket.IO 3.6. Cada servidor de corrección se suscribe al canal hasta que se le es otorgada una tarea de corrección, momento en el cual elimina su suscripción. Cuando dicho servidor termina la corrección, chequea que no existan tareas de corrección. En caso de que exista alguna la toma para corrección y ejecuta dicho proceso. Si no existen tareas de corrección pendientes, se suscribe de nuevo al canal.

El proceso ocurre de la siguiente manera:

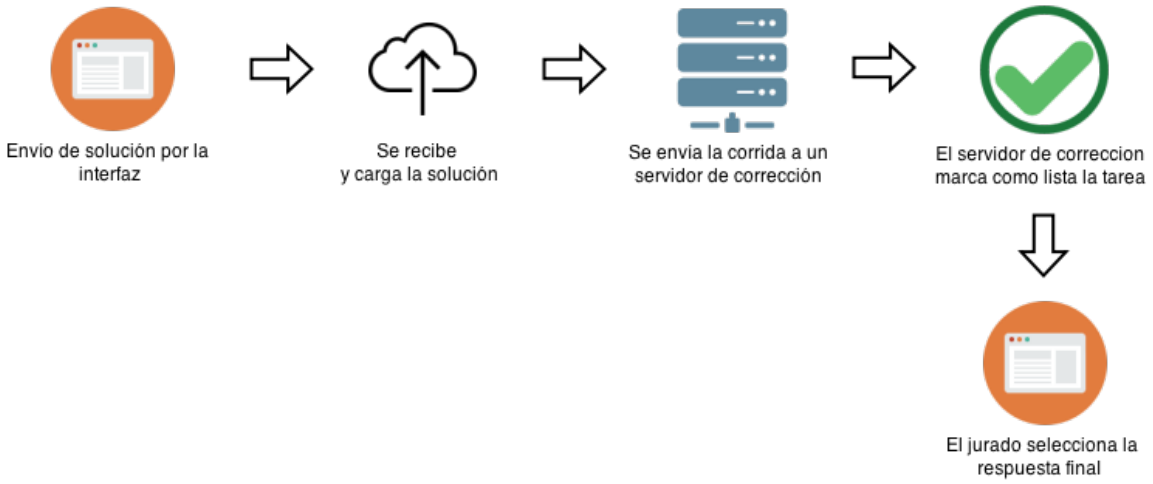

Figura 5.16: Diagrama Ciclo de vida de una solución

1. Un competidor o jurado decide enviar una solución, y para hacerlo hace uso de la interfaz de envío de soluciones.

2. El controlador *RunController* maneja la solicitud a través de la función *submit*.

3. Dicha función se encarga de validar que el usuario que envía la solución, esté autorizado para enviar una solución para ese problema en particular:

   (a) El usuario debe estar registrado para la competencia a la que pertenece el problema o tarea.

   (b) La competencia debe estar activa.

4. El código fuente de la solución se almacena en el servidor y se guarda la corrida en la base de datos.

5. Se invoca a la función *dispatchRun* del servicio *JudgeService*.

6. La función *dispatchRun* obtiene los servidores de corrección que se encuentran conectados y esperando por una solución.

   (a) En caso de no haber servidores de corrección, se guarda la corrida con estatus "pendiente".

   (b) Si hay uno o más servidores de corrección disponibles, se completa la información necesaria para la corrección de la solución usando la función *fillGradeData* del servicio *GradeService*, y se le entrega la corrida al primer servidor de corrección disponible en la lista.

7. Luego de que es invocado el proceso de Corrección, el ciclo de vida continúa cuando el servidor de corrección asignado termina la tarea y la guarda con estatus "listo".

8. Al recibir la respuesta del servidor de corrección, la misma es mostrada en la interfaz del jurado.

9. Luego de que el jurado ha seleccionado la respuesta que desea mostrar, se actualiza el marcador de ser necesario y la interfaz del usuario para reflejar la respuesta obtenida.

10. Termina el proceso.

5.5.6. Rejuzgar una solución

A menudo en competencias de programación, se hace necesario hacer actualizaciones o cambios a los casos de prueba de los problemas. Luego de hacer esto, para mantener la consistencia en los resultados, el jurado puede optar por realizar un rejuzgado de los problemas alterados, lo que consiste en generar respuestas a través de la corrección de cada una de las corridas enviadas para dicho problema. Esto se maneja de la siguiente manera en el sistema:

1. El jurado selecciona el problema que desea rejuzgar y realiza dicha acción en la interfaz.

2. Se eliminan del cache del marcador todas las filas referentes al problema a rejuzgar.

3. Se vuelven a insertar las filas anteriores, pero esta vez sin información acerca del puntaje.

4. Por cada corrida enviada para el problema en cuestión, se invoca a la función *dispatchRun* del servicio *JudgeService*.

5. Se continúa con el ciclo de vida de una solución 5.5.5 a partir del punto *6*.

5.5.7. Globos

El sistema permite el manejo de la entrega de globos que se estila en algunos estándares de competencia como el ACM-ICPC 1.1. Este proceso ocurre como sigue:

1. Al culminar el ciclo de vida de una solución, si dicha solución es correcta para todas las subtareas del problema, y todas las subtareas del problema están configuradas para mostrar el resultado total, se crea un objeto *Balloon* que es almacenado en la base de datos.

2. Los objetos *Balloon* son mostrados en la interfaz del personal en tiempo real.

3. Cuando ya se ha entregado el globo al equipo o participante, se utiliza la interfaz para marcar la tarea como lista y se termina el proceso.

5.5.8. Solicitud de atención del personal

En caso de que un equipo necesite algún tipo de asistencia por parte del personal, se provee esta funcionalidad dentro del sistema en forma de un botón en la interfaz del competidor. El ciclo de vida de esta funcionalidad se describe a continuación:

1. El usuario hace uso del botón para llamar al personal. Se guarda un nuevo objeto *SOS* en la base de datos.

2. En la interfaz del personal se muestran las llamadas en tiempo real. Aparece una nueva llamada.

3. Una vez que la llamada ha sido atendida, se utiliza la interfaz para marcar la tarea como lista y se termina el proceso.

5.5.9. Impresiones

En algunos tipos de competencia se permite a los competidores imprimir sus códigos fuente para mejor análisis. Para lograr dicha funcionalidad, se desarrolló un módulo en donde los competidores pueden cargar su código fuente y que luego será impreso por el personal utilizando la utilidad de impresión del navegador. Esto ocurre de la siguiente manera:

1. El competidor carga su código fuente utilizando la interfaz provista para esta tarea. Se guarda esta tarea de impresión en la base de datos y es representada por un objeto *Print*.

2. En la interfaz del personal se muestran las tareas de impresión en tiempo real.

3. El personal puede hacer click en el botón que se muestra junto a cada tarea para ver el código fuente.

4. En esta nueva vista se presenta el código fuente, y un botón para imprimir dicho código.

5. Luego de que dicho código es impreso y entregado al equipo o competidor correspondiente, se marca como lista la tarea a través de interfaz y se da por culminado el proceso.

5.5.10. Solicitud de aclaración

En la mayoría de las competencias presenciales, se permite a los competidores hacer preguntas que deberán ser respondidas por miembros del jurado. Ese intercambio de información se maneja de la siguiente manera dentro del sistema:

1. Un competidor utiliza la interfaz provista para realizar una solicitud de aclaración.
2. El jurado recibe dicha solicitud y se dispone a responderla.
3. En este punto el jurado puede enviar la respuesta solo al competidor o equipo que la realizó, ó puede enviarla a todos los competidores.
4. El proceso culmina.

5.6. Frontend e Interfaz

5.6.1. Frontend

El código que compone a la aplicación del lado del cliente se encuentra escrito en el lenguaje de programación Javascript 3.3.1, utilizando el framework AngularJS 3.5 en conjunto con la biblioteca Socket.IO 3.6, en específico la provista por el framework SailsJS 3.5 que implementa algunas funciones de utilidad sobre el API de Socket.IO.

Para mantener la estructura de archivos y carpetas propias de AngularJS, optamos por seguir un enfoque modular, separando los archivos de Javascript por módulos, según los roles de usuario, y guardandolos en carpetas individuales. Esto siguiendo las buenas prácticas descritas en este artículo [54].

AngularJS es una herramienta bastante poderosa a la hora de desarrollar aplicaciones de una sola página, lo cual aprovechamos para nuestro sistema en conjunto con el módulo *UI-Router*, que facilita el diseño de las rutas dentro de la aplicación.

En este sistemas nos aprovechamos de la funcionalidad que posee AngularJS, la unión de datos en ambas vías. Esto en conjunto con los eventos manejados por Socket.IO, constituyen la mayor parte del manejo de los datos en el código del lado del cliente. En consecuencia de esta estructura, los datos críticos, son actualizados en tiempo real en la interfaz, haciendo así que el uso de la aplicación sea bastante fluido.

5.6.2. Interfaz

La interfaz se construyó en su totalidad utlizando la biblioteca Bootstrap 3.6. Esta herramienta provee un sistema de cuadrícula que facilitó la construcción de las diferentes interfaces y la organización de la misma. Se utilizaron diversas funcionalidades de HTML en su versión 5 y CSS3 para construir una interfaz fluida y agradable al usar. Se construyó una fuente de iconos hecha a la medida para el sistema, esto para agregar ayudas visuales para las diferentes secciones a la vez que se ahorra ancho de banda, al ser una fuente hecha a la medida de menor tamaño.

Básicamente la interfaz está compuesta por 4 módulos:

- Una cabecera con el logo y nombre del sistema, una sección donde se muestra información de las competencias próximas y en ejecución, y la sección clásica de acceso de usuarios donde, en caso de estar identificado dentro del sistema, se proveen vínculos a las acciones básicas de cada usuario, como son el cambio de contraseña y la salida del sistema. En caso de no estar identificado esta sección muestra el botón de ingreso al sistema.

- Si se está identificado dentro del sistema, se muestra un sistema de navegación en forma de pastillas, que contiene vínculos a las secciones pertinentes a cada tipo de usuario según su rol. En caso de existir algún tipo de evento dentro de esas secciones, las mismas pastillas reflejaran un contador de eventos.

- El contenido de la sección actual donde se encuentre el usuario.

- Un pié de página bastante sencillo con el nombre del sistema.

El tema y el esquema de colores utilizado en la interfaz es el tema por defecto de la herramienta Bootstrap.

5.7. Seguridad

Esta sección tiene como propósito la exposición del esquema de seguridad que se usa en la comunicación de los datos de la aplicación. La comunicación entre el servidor principal y el resto de componentes del sistema, se realiza a través de solicitudes HTTP y Websockets. Para asegurar dichas solicitudes, se utilizan los siguientes esquemas:

5.7.1. Autenticación

Para el manejo de la autenticación, se utilizaron JSON Web Tokens 3.1, de manera de evitar el manejo de sesiones del lado del servidor, disminuyendo la cantidad de memoria utilizada, e implementar un sistema de autenticación en el cual no sean necesarias cookies y mantener la arquitectura orientada a servicios dentro del servidor principal.

5.7.2. Autorización

En cuanto a la autorización, se implementó un sistema de acceso en base a roles, siendo los roles una analogía de los roles existentes ya en la organización de un maratón de programación, descritos en la sección Roles 5.3. Para imponer dicho esquema de autorización se desarrollaron políticas de acceso cuya base es provista por SailsJS 3.5. Dichas políticas se encargan de chequear que exista un token en la solicitud recibida, y revisar luego el rol del usuario que realiza dicha solicitud para decidir si puede acceder a la acción que intenta realizar.

5.7.3. Directorios Protegidos

Para proteger información sensible para la competencia, como los códigos fuentes de los competidores, entradas y salidas de los casos de pruebas, y las impresiones; se implementó un sistema de seguridad a nivel de directorios para proteger a dichos archivos de usuarios maliciosos.

En específico, se crearon dos directorios:

- *Protected* en donde se almacenan los archivos de entrada y salida de los casos de prueba, además de los códigos fuente enviados por los competidores como solución. Este directorio solo puede ser accedido por los Jurados y los Servidores de Corrección.
- *Jobs* en este directorio se almacenan los archivos enviados por los competidores para su impresión. Los archivos dentro de este directorio solo pueden ser leídos por el personal.

El acceso a estos directorios se realiza mediante un controlador y son protegidos mediante el uso de las políticas descritas anteriormente.

Capítulo 6

Pruebas

En este capítulo se exponen las diferentes pruebas realizadas al sistema. Todas las pruebas realizadas han sido de caja negra[55], y las hemos clasificado en pruebas de rendimiento y pruebas de aceptación.

6.1. Pruebas de Rendimiento

6.1.1. Carga de peticiones HTTP

Esta prueba consiste en el envío de peticiones HTTP hacia la dirección principal de la aplicación para medir los tiempos de respuesta que ofrece la aplicación ante cierta cantidad de solicitudes. Para hacer la prueba se utilizó la herramienta *Apache Benchmark*[56], la cual, permite el envío de peticiones HTTP al servidor web de la aplicación.

Para esta prueba seleccionamos como número de peticiones que se iban a enviar, los siguientes valores $N_0 = 10000, N_1 = 100000, N_2 = 1000000$.

Con $N_0 = 10000$ se puede observar en la figura 6.1.1, que los resultados son positivos, casi todas las solicitudes fueron atendidas en menos de $20ms$.

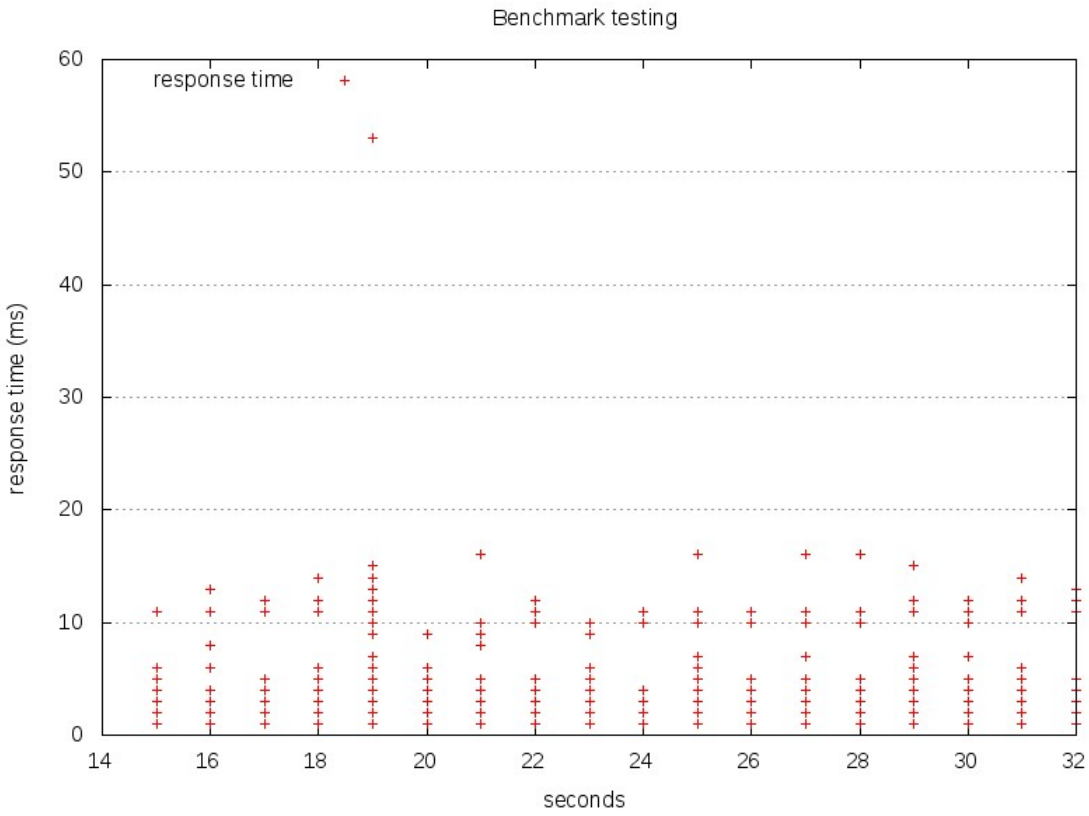

Figura 6.1: Carga de peticiones HTTP $N_0 = 10000$

Con $N_1 = 100000$ se puede observar en la figura 6.1.1, de igual manera que en el caso anterior, tenemos resultados positivos, permitiendo atender las solicitudes en un tiempo de respuesta menor a los $20ms$ la mayor parte de las solicitudes.

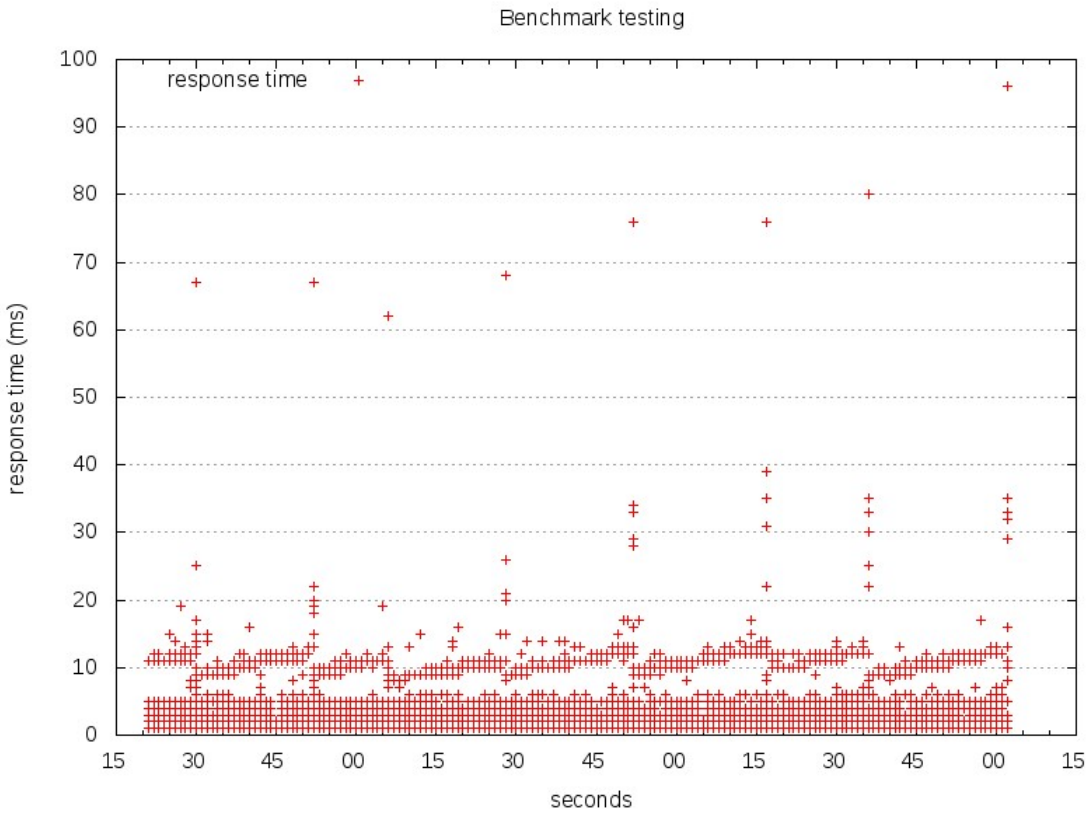

Figura 6.2: Carga de peticiones HTTP $N_1 = 100000$

Continuando con $N_2 = 1000000$ se puede observar en la figura 6.1.1, al igual que en los casos anteriores, tenemos resultados positivos, permitiendo responder la gran mayoría de las solicitudes en un márgen de $50ms$, ofreciendo buenos tiempos de respuesta aún cuando el número de peticiones aumenta considerablemente.

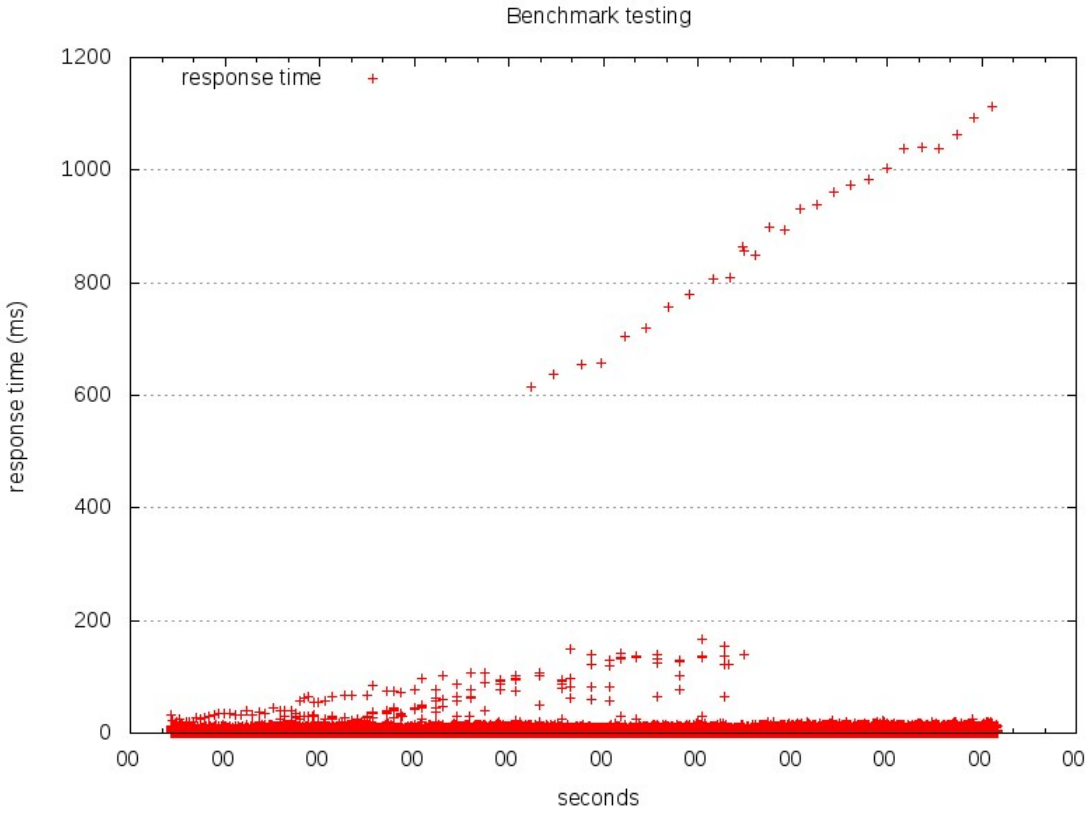

Figura 6.3: Carga de peticiones HTTP $N_2 = 1000000$

Luego de observar los resultados de las gráficas anteriores, podemos señalar que para una aplicación como la que estamos desarrollando, con el objetivo específico de albergar competencias de programación presenciales principalmente, donde suelen haber en nuestra experiencia un máximo de 500 competidores, y alrededor de 5000 a 10000 personas externas para consultar resultados, el hecho de puede atender 1000000 de peticiones en un tiempo menor a los $50ms$ puede considerarse bastante bueno, lo cual, nos permite afirmar que la aplicación va a ser percibida con muy buenos tiempos de respuesta por los usuarios, ayudando a cumplir el objetivo de proveer una experiencia fluida durante la competencia.

6.1.2. Carga de trabajo procesando soluciones

En esta prueba el juez tiene 200 soluciones enviadas por los competidores, pendientes por ser juzgadas, el objetivo es medir el tiempo que se demora en procesar dichas soluciones la aplicación, utilizando uno o varios correctores para encargarse del trabajo. Específicamente la prueba fue realizada utilizando $N = [1, 2, 4]$ daemons.

Como se puede observar en la figura 6.1.2, el tiempo que toma al juez procesar las solicitudes es inversamente proporcional a la cantidad de daemons disponibles para la corrección. Con un solo daemon trabajando puede tomar cerca de $1000s$ procesar todos los 200 problemas, sin embargo, al elevar el número de daemons, se reduce el tiempo a cerca de unos $200s$.

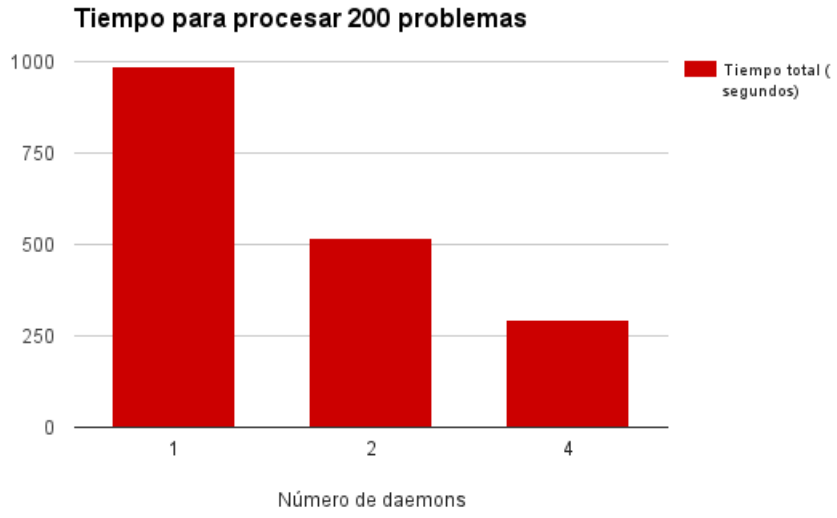

Figura 6.4: Tiempo necesario para procesar 200 solicitudes según el número de daemons

Durante la prueba se pudo observar que el consumo de memoria y procesador aumenta a medida que se aumenta la cantidad de daemons para la corrección, sin embargo, no excede a los $50MB$ adicionales en el caso de utilizar los 4 daemons en la

misma máquina, igualmente el agregar varios daemons, permite un uso eficiente de los procesadores de múltiples núcleos, dado que se puede asignar un corrector por núcleo del procesador. Se pudo apreciar también que el tiempo promedio que toma corregir un problema por parte de los daemons, se mantiene relativamente constante en torno a los $4 - 6s$. Adicionalmente debemos hacer mención a que el tiempo de la corrección puede ser afectado por los tiempo límite de cada problema, además de la naturaleza que pueda tener el mismo.

En esta prueba podemos ver que la tecnología de *Docker* [53] nos ofrece un muy buen rendimiento para la gestión del ambiente controlado, y el uso de los recursos disponibles de forma rápida y sencilla.

6.1.3. Carga de conexiones vía Web Sockets

En esta sección nos apoyamos en estudios realizados sobre Socket.IO 3.6 por Mikito Takada [59], donde podemos apreciar las capacidades para manejar conexiones concurrentes que posee.

La prueba que fue realizada consiste en realizar conexiones simultáneas por sockets y enviar un mensaje para que el servidor lo responda hasta alcanzar los $5s$ de uso del CPU. Para poder realizar una nueva conexión, tiene que haberse establecido la conexión anterior y así sucesivamente. El objetivo de la prueba es descubrir cuántas conexiones por sockets es capaz de manejar el servidor durante los $5s$ de tiempo de uso de CPU.

Los resultados arrojaron que utilizando Socket.IO 0.8.6 el servidor es capaz de manejar aproximadamente 1900 conexiones abiertas en el peor de los casos, todo ello utilizando un solo servidor con un solo núcleo, lo cual, nos permite cubrir la demanda de usuarios para la cual está siendo diseñada la aplicación.

6.2. Pruebas de Aceptación

En esta sección presentamos una evaluación desde el punto de vista de los usuarios de la aplicación. Se realizó una competencia piloto, con competidores de todos los rangos, experimentados, intermedios y nuevos, utilizando un conjunto de problemas variado para que todos pudieran resolverlo, con el objetivo de que los usuarios pudieran ver la aplicación y compararla con las que se vienen utilizando actualmente.

Para realizar la evaluación se realizó una encuesta sencilla, donde los competidores debían responder mediante una selección simple en escala del 1 al 5, siendo el 1 la calificación deficiente y 5 la sobresaliente. A continuación presentamos los resultados de la evaluación:

- ¿Cómo evalua el tiempo de respuesta de la aplicación? En esta pregunta la mayor parte de los encuestados, dio una valoración positiva, todas las respuestas estuvieron entre 3 y 5 puntos, indicando que los tiempos de espera fueron bastante bajos durante el transcurso de la competencia.

- ¿Le resultó fácil utilizar la aplicación? En esta pregunta el resultado fue muy bueno, todas las respuestas estuvieron entre 4 y 5 puntos, en su mayoría 5 puntos, indicando que a los usuarios les pareció bastante intuitiva la aplicación.

- ¿Le pareció agradable el diseño? Aquí la aplicación se destacó, obteniendo un 100 % de respuestas de 5 puntos, indicando que a los participantes les gusto mucho el diseño.

- ¿Le pareció estable el funcionamiento de la aplicación? Aquí la aplicación tuvo ciertos detalles durante la competencia, propios de una versión beta, sin embargo, en líneas generales tuvo una valoración positiva en cuanto a la estabilidad, teniendo 1 solo voto por debajo de los 3 puntos y el restante de votos en su mayoría con 4 puntos, con lo cual, podemos presumir un comportamiento bastante bueno.

En la figura 6.2 podemos observar los resultados de las preguntas de forma gráfica. Lamentablemente a pesar de haber tenido 18 participantes, solamente 6 decidieron responder la encuesta, y es en base a este universo que determinamos los resultados.

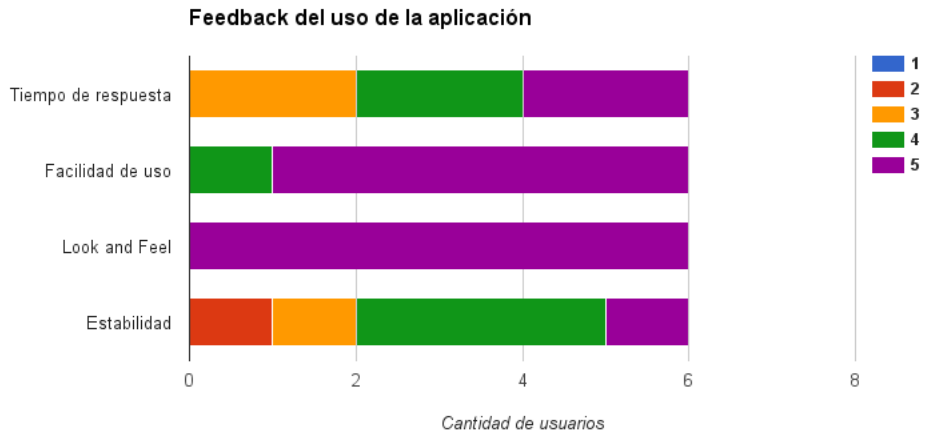

Figura 6.5: Resultados de las pruebas de aceptación por parte de los usuarios

Otra pregunta que se realizó durante la encuesta era para ver si los participantes estarían interesados en aportar desarrollos futuros al proyecto en caso de volverlo código abierto, la respuesta fue positiva y varios de ellos afirmaron estar de acuerdo en colaborar con el proyecto.

Conclusiones

Las competencias de programación, son eventos importantes dentro del mundo de las ciencias de la computación. Anualmente son muchos los estudiantes y profesionales de todos los niveles los que participan en este tipo de competencias para medir sus conocimientos de algoritmos y programación.

A medida que han ido avanzando las tecnologías y las ciencias de la computación en general, también han ido aumentando en cantidad y en variedad las competencias de programación. Sin embargo, las plataformas utilizadas para gestionar dichas competencias no han evolucionado de la misma manera, haciendo complicado para los organizadores el poder gestionarlas.

La variedad de reglas y formatos, han hecho que muchas aplicaciones para gestionar las competencias, se enfoquen solamente en las más importantes como la ACM-ICPC o la Olimpíada de Informática.

Con la aplicación que hemos desarrollado, es posible gestionar un rango más amplio de competencias, permitiendo a través de configuraciones sencillas, administrar competencias de diferentes tipos, reglas y formatos. La arquitectura distribuida del sistema permite descargar del trabajo de corrección al servidor principal, mejorando los tiempos de respuesta para los usuarios.

Adicionalmente hemos logrado proveer una herramienta con avances tecnológicos importantes en comparación a las existentes, permitiendo una experiencia de usabilidad y desempeño muy agradable tanto para los competidores como para los jurados y miembros del personal.

Con las pruebas realizadas hemos podido constatar un buen rendimiento de la aplicación bajo condiciones de estrés, y tiempos de respuesta. En el área de usabilidad los usuarios de la competencia piloto nos reportaron excelentes resultados.

Sin embargo, a pesar de tener muy buenas impresiones en cuanto al desempeño de la aplicación durante las pruebas y la competencia piloto que fue realizada, es necesario que la aplicación continue su proceso de maduración para ir ganando la confianza de los organizadores a nivel nacional, regional y por que no mundial.

Bibliografía

[1] http://icpc.baylor.edu/worldfinals/rules

[2] ICPC Excecutive Comitee. Policies and Procedures for the ACM International Collegiate Programming Contest. 2001. http://icpc.baylor.edu/download/compete/pdf/ICPC-Policies-and-Procedures.pdf

[3] http://icpc.baylor.edu/compete/problems

[4] http://icpc.baylor.edu/regionals/rules

[5] http://icpc.baylor.edu/download/regionals/rules/EligibilityDecisionTree-2014.pdf

[6] Verhoeff T., Horváth G., Diks K., Cormack G., Forišek. The International Olympiad in Informatics Syllabus. 2013. http://ioinformatics.org/a_d_m/isc/iscdocuments/ioi-syllabus.pdf

[7] http://ioinformatics.org/history.shtml

[8] General Assembly. IOI 2010. http://ioinformatics.org/rules/reg10.pdf

[9] http://apps.topcoder.com/wiki/display/tc/Competing+in+a+Rated+Algorithm+Competition

[10] http://apps.topcoder.com/wiki/display/tc/Algorithm+Competition+Rating+System

[11] http://apps.topcoder.com/wiki/display/tc/How+to+Compete+in+SRM+Algorithm+Competitions

[12] http://codeforces.com/help

[13] http://codeforces.com/blog/entry/4088

[14] https://code.google.com/codejam/faq.html

[15] https://code.google.com/codejam/terms.html

[16] https://www.facebook.com/hackercup/terms

[17] https://www.facebook.com/notes/facebook-hacker-cup/hacker-cup-2013-faq/591459627536609

[18] PC ^ 2 Página Principal http://www.ecs.csus.edu/pc2/

[19] Kattis Página Principal. 2006. https://kattis.csc.kth.se/

[20] Hackzor proyecto en Google Code https://code.google.com/p/hackzor/

[21] Boca Online Contest Administrator Página Principal http://www.ime.usp.br/~cassio/boca/

[22] DOMjudge Página Principal http://www.domjudge.org/

[23] Midas Judge proyecto en Google Code https://code.google.com/p/midas-judge/

[24] U WP Judging Tool http://micsymposium.org/mics_2010_proceedings/mics2010_submission_7.pdf

[25] WACS http://cdn.intechopen.com/pdfs/8858/InTech-Smart_web_based_programming_contests_management_tool.pdf

[26] "Customer Information Manager (CIM)". SOAP API Documentation. Authorize.Net. Julio 2013.

[27] "What is Object/Relational Mapping?"http://hibernate.org/orm/what-is-an-orm/

[28] Object-Oriented Database (OODBMS) Free Resource Portal. ODBMS. Agosto 2013.

[29] Coulouris, George; Jean Dollimore; Tim Kindberg; Gordon Blair (2011). Distributed Systems: Concepts and Design (5th Edition). Boston: Addison-Wesley. ISBN 0-132-14301-1.

[30] "Distributed Application Architecture". Sun Microsystem. Junio 2009.

[31] "More deeply, the framework exists to separate the representation of information from user interaction."The DCI Architecture: A New Vision of Object-Oriented Programming - Trygve Reenskaug and James Coplien. Marzo 2009.

[32] Bader, David; Robert Pennington . "Cluster Computing: Applications". Georgia Tech College of Computing. Julio 2007.

[33] Bachman, Charles W.. «The programmer as navigator». Consultado el 17 febrero 2013.

[34] O'Reilly Media, Inc. Learning Python, Fourth Edition (libro). O'Reilly. Consultado el 11 de febrero de 2010.

[35] http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf

[36] M. Domínguez-Dorado,. Todo Programación. Nº 12. Págs. 48-51. Editorial Iberprensa(Madrid). DL M-13679-2004. Septiembre, 2005. Bases de datos en el cliente con JavaScript DB.

[37] http://es.wikipedia.org/wiki/Lenguaje_de_marcado

[38] http://es.wikipedia.org/wiki/HTML

[39] http://es.wikipedia.org/wiki/Hojas_de_estilo_en_cascada

[40] Riehle, Dirk (2000), Framework Design: A Role Modeling Approach, Swiss Federal Institute of Technology

[41] http://en.wikipedia.org/wiki/Push_technology

[42] Krill, Paul (September 24, 2007). "AJAX alliance recognizes mashups". InfoWorld. Consultado 2010-10-20.

[43] http://en.wikipedia.org/wiki/WebSocket

[44] Sitio Web de NodeJS http://nodejs.org/

[45] http://benchmarksgame.alioth.debian.org/u64/benchmark.php?test=all&lang=v8&lang2=python3

[46] Sitio Web de SailsJS http://sailsjs.org/

[47] http://en.wikipedia.org/wiki/Socket.IO

[48] http://mj.ucw.cz/papers/isolate.pdf

[49] Martin, Robert C.. "Design Principles and Design Patterns". Consultado en 2000. http://www.objectmentor.com/resources/articles/Principles_and_Patterns.pdf

[50] http://en.wikipedia.org/wiki/Publish%E2%80%93subscribe_pattern

[51] Flanagan, David, "JavaScript - The Definitive Guide", 5th ed., O'Reilly, Sebastopol, CA, 2006, p.497

[52] "What Is Angular?"http://docs.angularjs.org/guide/introduction. Consultado el 12 February 2013.

[53] "What is Docker?". Consultado el 4 de marzo de 2015 https://www.docker.com/whatisdocker/

[54] "AngularJS Best Practices: Directory Structure". Consultado el 19 de marzo de 2015 https://scotch.io/tutorials/angularjs-best-practices-directory-structure

[55] Boris Beizer., "Black-Box Testing: Techniques for Functional Testing of Software and Systems.", 1995., John Wiley & Sons, Inc., New York NY, USA

[56] "Apache Benchmark Tool". Consultado el 20 de marzo de 2015 http://httpd.apache.org/docs/2.2/programs/ab.html

[57] "The Ins and Outs of Token Based Authentication". Consultado el 22 de marzo de 2015 https://scotch.io/tutorials/the-ins-and-outs-of-token-based-authentication

[58] "The Anatomy of a JSON Web Token". Consultado el 22 de marzo de 2015 https://scotch.io/tutorials/the-anatomy-of-a-json-web-token

[59] "Performance Benchmarking Socket.IO". Consultado el 10 de abril de 2015. Mikito Takada. http://blog.mixu.net/2011/11/22/performance-benchmarking-socket-io-0-8-7-0-7-11-and-0-6-17-and-nodes-native-tcp/