ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

PROTOTIPO DE SISTEMA DISTRIBUIDO PARA TORNEOS DE PROGRAMACIÓN BASADO EN SERVICIOS WEB DE TIPO REST

TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN ELECTRÓNICA Y REDES DE INFORMACIÓN

DAVID EDISON NÚÑEZ QUIJIJE
david.nunez.dev@gmail.com

DIRECTOR: ING. RAÚL DAVID MEJÍA NAVARRETE M.Sc.
david.mejia@epn.edu.ec

Quito, septiembre 2018

AVAL

Certifico que el presente trabajo de titulación fue desarrollado por David Edison Núñez Quijije, bajo mi supervisión.

Ing. David Mejía MSc.
DIRECTOR DEL TRABAJO DE TITULACIÓN

DECLARACIÓN DE AUTORÍA

Yo David Edison Núñez Quijije, declaro bajo juramento que el trabajo aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración cedo mis derechos de propiedad intelectual correspondientes a este trabajo, a la Escuela Politécnica Nacional, según lo establecido por la Ley de Propiedad Intelectual, por su Reglamento y por la normatividad institucional vigente.

David Edison Núñez Quijije

AGRADECIMIENTO

A mis padres Gabriel y Lorena; mis hermanos Gabriel y Paola por tanta paciencia y todo ese apoyo incondicional que han sabido brindarme en todos los aspectos buenos y malos de mi vida.

A mis abuelitos Gabriel, Zoila, Olinda y Pablo, porque han sabido confiar y esperar con paciencia para que pueda concluir con mis estudios.

A mi director de Proyecto de Titulación M.Sc. David Mejía por el apoyo, paciencia y las oportunidades brindadas para poder finalizar este trabajo.

A esos grandes genios de las ciencias de la computación y la comunidad Open-Source pues, han sido mi motivación para estar involucrado con las últimas tecnologías de software. Gracias a ellos, tengo una pasión por las ciencias de la computación.

DEDICATORIA

A mis padres, hermanos y abuelitos. Gracias por su paciencia y confianza de que algún día este día llegaría.

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE CÓDIGOS

ÍNDICE DE ECUACIONES

ÍNDICE DE TABLAS

RESUMEN

El presente Trabajo de Titulación se centra en el desarrollo de un prototipo de sistema distribuido para torneos de programación.

En el primer capítulo, se presenta una perspectiva general de los fundamentos enfocados a desarrollar el prototipo propuesto. Se tratan algunos temas fundamentales relacionados con arquitectura de software, modularización basado en el patrón MVC (Modelo Vista Controlador), tecnología de contenedores Docker, definición de componentes web, *WebSocket* para proporcionar una comunicación asincrónica y *full-duplex* entre cliente y servidor, *frameworks*, librerías, servicios en la nube y la metodología de gestión del proceso para el desarrollo de software.

En el segundo capítulo, se presenta un breve análisis del proceso seguido para diseñar, desarrollar y desplegar los componentes que son requeridos. En primer lugar, se analiza la plataforma web CodeFights para obtener una perspectiva sobre las aplicaciones web de torneos de programación. A continuación, se obtienen algunos requisitos funcionales y no funcionales. Teniendo en cuenta los requisitos, se realizan diagramas UML para obtener una perspectiva del prototipo. Luego, se detalla el proceso de desarrollo de cada componente y utilidades para el *front-end* y el *back-end*. Después, se emplea Docker para construir una imagen que sirve como contenedor de aplicaciones. Finalmente, se muestra el proceso de despliegue del prototipo utilizando los servicios de AWS (*Amazon Web Services*).

El tercer capítulo presenta los resultados de las pruebas realizadas sobre el prototipo desarrollado ya desplegado en AWS.

Finalmente, en el cuarto capítulo, se presentan conclusiones y recomendaciones, basadas en todo el proceso llevado a cabo para obtener el prototipo.

PALABRAS CLAVE: WebSocket, Docker, AWS, CodeFights, MVC.

ABSTRACT

The current project is focused on developing a distributed system prototype for coding tournaments.

In the first chapter, a general perspective of the fundamentals about concepts focused to develop this proposed prototype is given. Topics related with software architecture, modularization based on MVC (Model View Controller) pattern, Docker containers technology, WebSocket to provide an asynchronous and full-duplex communication between client and server, frameworks, libraries, cloud services, and methodologies to management the process to develop the software are treated.

The second chapter, is focused in analyzing the process to design, develop and deploy the required components. First of all, the CodeFights web platform is analyzed to get a perspective about tournament web applications. After that, some functional and non-functional requirements for this prototype are obtained. Having in mind the requirements, UML diagrams are done to get a perspective about this prototype. Then, the process to develop each component and utilities for front-end and back-end is detailed. Next, a Docker for building an imagen to containerize the system is used. Finally, the process to deploy the prototype using AWS (Amazon Web Services) services is shown.

The third chapter shows tests outcomes of the prototype developed when it is deployed on AWS.

Finally, the fourth chapter includes conclusions and recommendations based on the whole process carried out to get the prototype.

KEYWORDS: WebSocket, Docker, AWS, CodeFights, MVC.

1 INTRODUCCIÓN

Según [1] la programación competitiva es el proceso de resolver, tan rápido como sea posible, problemas con algoritmos bien conocidos y, que previamente han sido resueltos por los proponentes de un torneo de programación. En dichos torneos, a los concursantes se les da un conjunto de problemas con el propósito de que escriban sus propios algoritmos de tal manera que puedan resolver los problemas propuestos.

Para realizar la calificación de los algoritmos que son enviados por los competidores, generalmente se cuenta con un sistema computacional llamado juez en línea, el cual se encarga no solo de compilar y ejecutar el código enviado por los competidores sino también, de evaluar, bajo ciertas restricciones, la solución propuesta. Dichas restricciones vienen dadas por la cantidad de memoria límite que se debe usar, el tiempo de ejecución, el límite de uso de procesamiento, entre otros. Estos sistemas tienen mecanismos de calificación basados en la rapidez de ejecución y bajo consumo de memoria de los algoritmos propuestos. Sin embargo, esto queda determinado por cada sistema juez en línea [2].

Este tipo de torneos no solo tienen la intención de promover la búsqueda de programadores que poseen la habilidad de resolver problemas con algoritmos, sino también, de estimular el trabajo en equipo e incentivar un aprendizaje y mejora de las habilidades en la resolución de algoritmos, basado en la competición y colaboración y que, de hecho, resulta ser mucho más dinámico, entretenido, menos tradicional y más desafiante [3], [4].

Por tales motivos, este tipo de eventos se llevan a cabo anualmente alrededor del mundo; son organizados por grandes compañías tales como Google con su programa Google Code Jam, Facebook con su programa Facebook Hacker Cup. O incluso, eventos que se desarrollan con mayor frecuencia a través de plataformas en línea como HackerRank, CodeCombat, CodeFights, entre otras.

Asimismo, entidades educativas, en asociación con la ACM ICPC (*Association For Computing Machinery International Collegiate Programming Contest*) llevan a cabo la competencia más grande del mundo y que es auspiciada por IBM. En cada competición, dichas entidades cuentan con un sistema juez en línea para evaluar el código enviado por sus competidores [3].

El Departamento de Electrónica, Telecomunicaciones y Redes de Información (DETRI) ha estado interesado en llevar a cabo torneos de programación para fomentar un mecanismo de aprendizaje complementario al tradicional, e incentivar a los estudiantes a que puedan demostrar sus habilidades en la resolución de problemas con algoritmos a través de dichos eventos.

Desde el 2015 hasta el 2017, se han realizado 3 concursos de programación y seguridad de TIC (Tecnologías de la Información y Comunicación) a escala nacional. Particularmente, para los torneos de programación, el DETRI cuenta con una herramienta en Moodle[1], que ha sido utilizada para dichos eventos.

La herramienta utilizada es *Virtual Programming Lab* (VPL) la cual permite evaluar y calificar de forma automática los *scripts* que son enviados por los estudiantes. Sin embargo, la idea fundamental detrás de VPL no está en ser utilizada como una herramienta de torneos de programación sino más bien, está enfocada en ser utilizada como un apoyo para los docentes en la enseñanza y automatización de tareas y, para los estudiantes, como un soporte en el proceso de aprendizaje [5].

Basado en las premisas anteriormente mencionadas, en el presente Trabajo de Titulación, se implementa una plataforma para torneos de programación.

1.1 Objetivos

El objetivo general de este Trabajo de Titulación es desarrollar un prototipo de sistema distribuido para torneos de programación utilizando Servicios Web de tipo REST.

Los objetivos específicos de este Trabajo de Titulación son:

Analizar los requerimientos del prototipo para lo cual se analizará la plataforma web de torneos de programación CodeFights.

Diseñar los componentes que conforman el prototipo, estos son: la aplicación web, la base de datos y los Servicios Web.

Implementar los componentes que conforman el prototipo para lo cual se utilizarán los servicios de AWS.

Analizar los resultados de las pruebas para corregir posibles fallos.

[1] Moodle es una plataforma web de gestión de contenidos para aprendizaje en línea.

1.2 Alcance

El prototipo propuesto contará con los siguientes componentes: una base de datos, una aplicación web y dos Servicios Web.

Dada la naturaleza del prototipo, habrá dos tipos de comunicaciones: comunicaciones originadas desde el cliente hacia el servidor y desde el servidor hacia el cliente. Para ello se utilizarán tecnologías como *WebSockets* y AJAX (*Asynchronous* JavaScript *And* XML).

Para el desarrollo del prototipo se utilizará la metodología de desarrollo ágil Kanban. Kanban será utilizada para gestionar las tareas que se llevarán a cabo en todo el proceso de desarrollo del prototipo.

Se utilizará la base de datos MySQL, la cual permitirá almacenar información de los usuarios, de los torneos, de los resultados de los retos propuestos, etc.

Los servidores web, la API RESTful para los Servicios Web y el *back-end* de la aplicación web serán codificados en el lenguaje de programación Go. Cada Servicio Web realizará las tareas de compilación o interpretación, ejecución y evaluación del código de un lenguaje de programación y el resultado de la evaluación se retornará.

Para la implementación del *front-end* de la aplicación web se utilizará Javascript, CSS (*Cascading Style Sheets*) y HTML (*HyperText Markup Language*). En la aplicación web se definirá la interfaz de usuario, la cual permitirá interactuar con el Servicio Web y la base de datos.

1.3 Marco Teórico

En esta sección, se describen aspectos fundamentales referentes a las tecnologías, librerías, *frameworks*, lenguajes de programación, *scripting* y maquetado, utilizados para la implementación del prototipo propuesto. También se detallan aspectos importantes en cuanto al patrón de diseño Modelo-Vista-Controlador (MVC) y la modularización aplicada a sistemas web modernos, teniendo como base el patrón MVC.

Por otro lado, se analiza en los conceptos principales del uso de contenedores para empaquetar varios de los componentes desarrollados y que estos, puedan ser desplegados utilizando algunos servicios en la nube. Esto con lleva que también se describan características importantes de algunos servicios que serán utilizados para el despliegue del prototipo. Adicionalmente, se revisa aspectos fundamentales acerca de pruebas unitarias; que es el mecanismo utilizado para evaluar el código enviado por los competidores. Por lo tanto, también se describe el uso de herramientas para implementar

dichas pruebas tanto para Java como para C# pues éstos son los lenguajes que soporta el sistema para los torneos de programación. Dichos lenguajes han sido seleccionados dado que son parte del pénsum académico de la carrera de Ingeniería en Electrónica y Redes de Información de la Escuela Politécnica Nacional. Finalmente, se aborda el tema de la metodología de desarrollo ágil Kanban para gestionar las tareas que se realizan en todo el proceso de desarrollo.

1.3.1. Sistemas Distribuidos

Un sistema distribuido es un sistema que consta de múltiples computadoras autónomas y que están conectadas entre sí, de tal manera que tienen la capacidad de compartir diferentes tipos de recursos y capacidades de forma simultánea para cumplir con una tarea en particular y, mostrarse al usuario, como un único sistema. Cada computadora puede estar ubicada dentro de una misma red o incluso, tener una ubicación geográfica distinta. No obstante, tienen la capacidad de poder cooperar entre sí, a través del paso de mensajes, usando una red como lo es Internet [6].

La Figura 1.1, ilustra las partes básicas de un sistema distribuido considerando 3 máquinas.

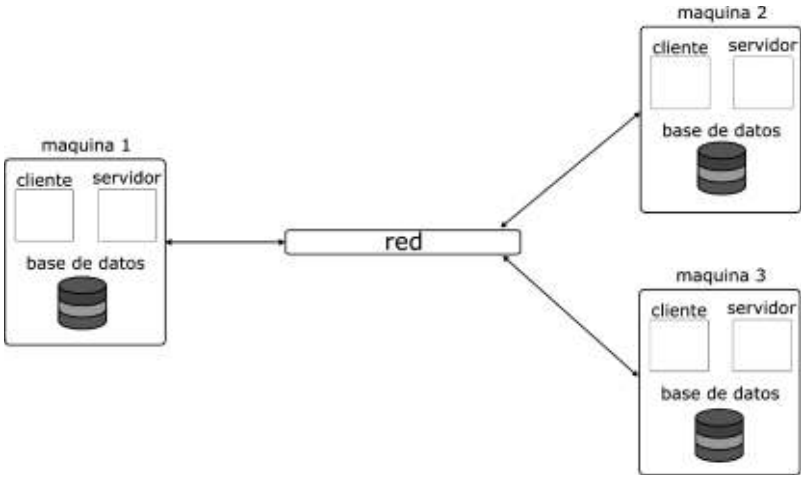


Figura 1.1 Sistemas distribuidos

1.3.2. El modelo Cliente – Servidor

Este modelo describe como se da la comunicación entre una aplicación cliente y una aplicación servidor y cómo éstas, pueden intercambiar información entre sí y, a través de la red (Figura 1.2).

El cliente es quién inicia la comunicación con el fin de solicitar un recurso o servicio mientras que el servidor se encarga de responder dicha solicitud.

Figura 1.2. Modelo Cliente - Servidor a través de la red

Cada componente tiene una separación de responsabilidades, que es el principio detrás de las restricciones cliente-servidor. Esta separación de responsabilidades permite simplificar el trabajo de cada componente mejorando así la escalabilidad y la evolución de cada uno de ellos de forma independiente siempre y cuando la interfaz entre las dos no cambie [7].

Mientras que una aplicación cliente, por lo general pero no obligatorio, cuenta con una interfaz gráfica, la aplicación servidor no cuenta con una, sin embargo, si provee un mecanismo que le permite exponer una API[2] (*Application Programming Interface*) para que aplicaciones clientes puedan realizar peticiones [8].

1.3.3. Aplicaciones web

Una aplicación web es una aplicación de software del tipo cliente-servidor que trabaja sobre el protocolo HTTP[3] (*HyperText Transfer Protocol*). Esta aplicación se encuentra alojada en un servidor web y se puede acceder a ella a través de Internet utilizando un navegador web. El navegador web es el encargado de presentar la interfaz gráfica de la aplicación web usando tecnologías como HTML[4], CSS[5] y Javascript[6]. La interfaz gráfica no es más que una página web que puede ser de tipo estática o dinámica. Cuando se habla de una página web estática, ésta no puede interactuar con el servidor y presenta contenido netamente estático, es decir, contenido que no cambia cada vez que se realiza una petición al servidor. Por otro lado, una página web dinámica si tiene la capacidad de

[2] Una API es una interfaz o punto de acceso de un componente de software que permite que otros componentes de software, previo acuerdo coordinado, puedan interactuar con dicha interfaz. El componente de software el cuál provee una API tiene un conjunto de funciones, técnicas y herramientas que permiten proveer un servicio o recurso al componente que lo ha solicitado [8]

[3] HTTP es un protocolo que permite transferir información hipermedia. Dicha información no es más que información de tipo texto para la web (Hipertexto), imágenes, videos, documentos de estilo, *scripts*, etc.[60]

[4] HTML es un lenguaje de marcado que permite definir una estructura, mediante etiquetas, a una página web.

[5] CSS es un lenguaje de maquetación que permite dar estilos de diseño a una página web. Las etiquetas de CSS están asociadas a un identificador único o general en cada etiqueta en el documento HTML.

[6] Javascript es un lenguaje *scripting* utilizado en el navegador para proveer, a una página web, mayores funcionalidades como validaciones de campos, animaciones, comunicación con el servidor por *background*.

interactuar con el servidor para solicitar y recibir un recurso o información haciendo que las páginas sean mucho más dinámicas [9].

a. Componentes Web

Los componentes web son bloques personalizados y re-utilizables que encapsulan etiquetas HTML, hojas de estilo CSS y Javascript, con el fin de extender la funcionalidad no solo de los elementos estándar que conforman el documento HTML, sino también de *frameworks* y librerías para el *front-end*. Estos componentes web son construidos siguiendo el estándar de componentes web base de la API HTML5[7] y trabajan en aquellos navegadores que soportan dicha versión [10].

b. AJAX

AJAX como tal, no es ninguna tecnología ni tampoco un protocolo o un lenguaje de programación para la web, sino más bien, es una técnica que usa un conjunto de tecnologías tales como HTML, Javascript, DHTML[8] (*Dynamic* HTML) y el *Document Object Model*[9] (DOM) para crear comunicaciones con el servidor de manera asíncrona - no bloqueante, *half-duplex* y bidireccional. Esto permite que, mientras el usuario continúa navegando sobre una página web, ciertas partes de dicha página puedan actualizar su contenido sin que esto implique una recarga completa de la página, creando así, páginas web mucho más dinámicas [11].

1.3.4. Representational State Transfer (REST)

REST es un estilo de arquitectura de ingeniería de software no estandarizado, híbrido y derivado de otros estilos de arquitectura basados en red, para sistemas hipermedia[10] distribuidos. La motivación detrás de REST, como lo menciona Roy Thomas Fielding, ha sido establecer una serie de restricciones arquitectónicas que, aplicadas en conjunto, definan la manera de cómo deberían crearse los servicios web modernos [12]. Estas son:

1. **Basado en el estilo Cliente – Servidor:** Debe existir una separación de responsabilidades tanto de la interfaz de usuario como en el almacenamiento de

[7] HTML5 es la última versión de HTML que permite agregar nuevas funcionalidades tales como creación de etiquetas personalidades, validación de formularios, entre otras.
[8] DHTML es una versión actualizada de la versión HTML el cual contiene etiquetas que permiten crear páginas web dinámicas las cuales permiten que haya una mayor interactividad entre el usuario y la página web [61].
[9] *Document Object Model* es una API para documentos HTML el cual define una descripción de interfaces y/o clases de dicho documento para que pueda ser accedido y manipulado utilizando Javascript a través de funciones.
[10] Hipermedia es una extensión de hipertexto, y es un término utilizado para especificar como elementos de tipo multimedia tales como: sonidos, videos, realidad virtual, imágenes, etc., pueden ser enlazados a otro contenido [11].

datos. Esto permitirá que mejore la portabilidad de la interfaz de usuario a múltiples plataformas, mejorando así también, la escalabilidad y simplicidad de los elementos que conforman el servidor.

2. **Sin estados:** Esta segunda restricción es definida para indicar que debe existir una limitación en la interacción entre el cliente y el servidor. Dicha limitación está enfocada en no mantener ningún estado en el servidor cuando un cliente está interactuando con este, sino que cada petición realizada por el cliente, debe contener absolutamente toda la información requerida para que dicha petición pueda ser entendida, procesada y, en consecuencia, pueda ser respondida.

3. **Caché:** Los datos que son enviados como una respuesta, previa solicitud, necesariamente deben ser etiquetadas como de tipo *cacheable* o no *cacheable*. Cuando los datos en una respuesta son etiquetados como *cacheable*, el cliente podrá reutilizar los datos de dicha respuesta para solicitudes posteriores.

4. **Interfaces uniformes:** Necesariamente debe existir un desacoplamiento entre el cliente y el servidor de tal manera que cada uno de ellos tenga la capacidad de poder evolucionar de forma independiente.

5. **Sistema de capas:** La idea fundamental de esta restricción es que el sistema permita ser presentado como una única entidad ante el cliente mientras que la arquitectura del servidor puede tener una jerarquía de capas para cada componente.

6. **Código bajo demanda (no obligatoria):** REST permite que la funcionalidad del cliente pueda temporalmente ser extendida a través de la descarga y ejecución de *scripts*.

1.3.5. Servicios Web RESTful

Un servicio web RESTful es un tipo de aplicación basado en la arquitectura REST y que trabaja sobre el protocolo HTTP. Los servicios web RESTFul no exponen una interfaz gráfica. En su lugar, proveen una API, sea ésta pública o privada, para responder a peticiones que son realizadas desde clientes externos ajenos al servicio web. El formato de intercambio de mensajes que se da entre el servicio web y los clientes puede ser JSON[11] (*Javascript Object Notation*), XML[12](*Extensible Markup Language*) u otro.

[11]JSON es un formato estándar basado en texto para representar estructuras de datos del tipo clave-valor. JSON, por ser ligero y menos verboso, es utilizado como una alternativa a otros formatos para el intercambio de información entre dos entidades [62].

Los servicios web RESTful mitigan la complejidad que representa la integración de varias aplicaciones permitiendo que, dicha integración, sea mucho más heterogénea, menos costoso y más escalable.

Estos beneficios se dan puesto que dichos servicios pueden ser desarrollados en cualquier lenguaje de programación, además, pueden ser desplegados en cualquier ambiente y plataforma pues la interacción que se dan entre dos entidades, únicamente está dada por el tipo de formato de intercambio de mensajes y el protocolo HTTP [7].

Hay un conjunto de principios que motivan a los servicios web RESTful a ser simples, ligeros y rápidos. Estos son [13]:

- **Identificación de recursos a través de una URI (*Uniform Resource Identifier*):** esto permitirá proporcionar un espacio de direccionamiento para el acceso a dicho recurso.
- **Interfaz uniforme:** los recursos son manipulados a través de un conjunto fijo de cuatro operaciones. Estas son: creación, lectura, actualización y eliminación. Dado que los servicios web RESTful trabajan sobre el protocolo HTTP, dichas operaciones son asociadas a los métodos de HTTP: GET[13], POST[14], PUT[15] y DELETE[16]. Ambas partes deben acordar los esquemas que describen los recursos que se intercambian y las formas de como procesarlos.
- **Mensajes autodescriptivos:** Los recursos que se intercambia entre el cliente y el servidor tiene una información descriptiva a través de metadatos. Dichos metadatos se usan, por ejemplo, para controlar el almacenamiento en caché, detectar errores de transmisión, control de acceso, entre otros.
- **Interacciones con estado a través de hipervínculos:** cada interacción con algún recurso existente en el servicio web es sin estado; es decir, cada solicitud e intercambio de datos debe ser independientes y deben contener toda la información necesaria en sus metadatos o incluso otras técnicas como el uso de *Cookies*[17] o campos ocultos en formularios.

---

[12] XML es un formato de marcas que definen un conjunto de reglas para codificar un documento.
[13] GET es un método HTTP que se utiliza para leer un recurso.
[14] POST es un método HTTP que se utiliza para crear un nuevo recurso.
[15] PUT es un método HTTP que se utiliza para actualizar total o parcialmente un recurso existente.
[16] DELETE es un método HTTP que se utiliza para borrar un recurso existente.
[17] *Cookies* son elementos utilizados en la web para almacenar información importante del usuario quién está interactuando con un servidor. Es comúnmente utilizado puesto que la comunicación a través del protocolo HTTP es sin estados.

1.3.6. Patrón de diseño Modelo Vista Controlador

El Modelo-Vista-Controlador también conocido como MVC, es un patrón de diseño básico que se utiliza para modularizar una aplicación web en tres diferentes capas a nivel de la aplicación de tal manera que permita definir responsabilidades en cada una ellas.

A la capa modelo le corresponden funcionalidades de procesamiento de la información y persistencia de dicha información, a través de la interacción con la base de datos.

El controlador está encargado de ser una capa intermedia para procesar las peticiones del usuario y, posteriormente, interactuar con la capa modelo y la capa vista. Se podría decir que es una capa que delega actividades a las otras capas y depende de estas, para poder responder a la solicitud realizada por el usuario.

Finalmente, la vista está encargada de generar la interfaz final de usuario con la información que le es provista desde el controlador [14], [15].

En la Figura 1.3, se tiene una representación básica de la interacción de cada una de las capas que corresponden al patrón MVC.

Como se puede visualizar, el controlador es el intermediario entre el navegador y la capa modelo y vista. Sin embargo, es únicamente una representación básica general.

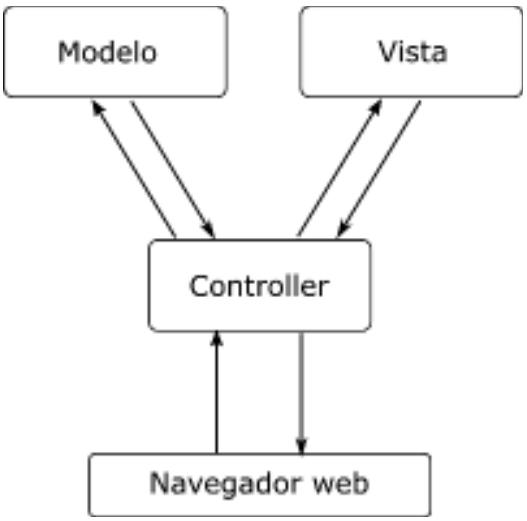


Figura 1.3. Esquema básico del Modelo Vista Controlador

En la Figura 1.4, se tiene un flujo de datos donde la capa modelo no tienen ninguna interacción. En otras palabras, la capa controladora no delega ninguna operación a la capa modelo y, por lo tanto, tampoco hay una interacción con la base de datos. Este proceso generalmente se da cuando el resultado final únicamente depende de contenido estático.

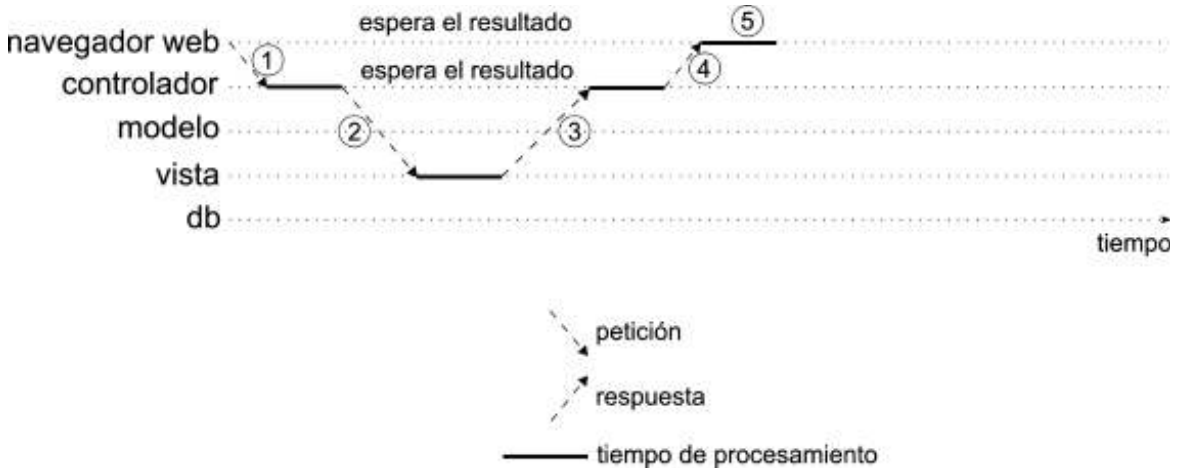


Figura 1.4. MVC – Línea de tiempo del flujo de datos donde actúan dos capas

En cambio, como se puede ver en la Figura 1.5, la interacción se da prácticamente en todas las capas puesto que la información que debe ser presentada al usuario que la solicitó, está estrechamente ligado con contenido que generalmente cambia y debe ser persistente, esto quiere decir, información almacenada en una base de datos.

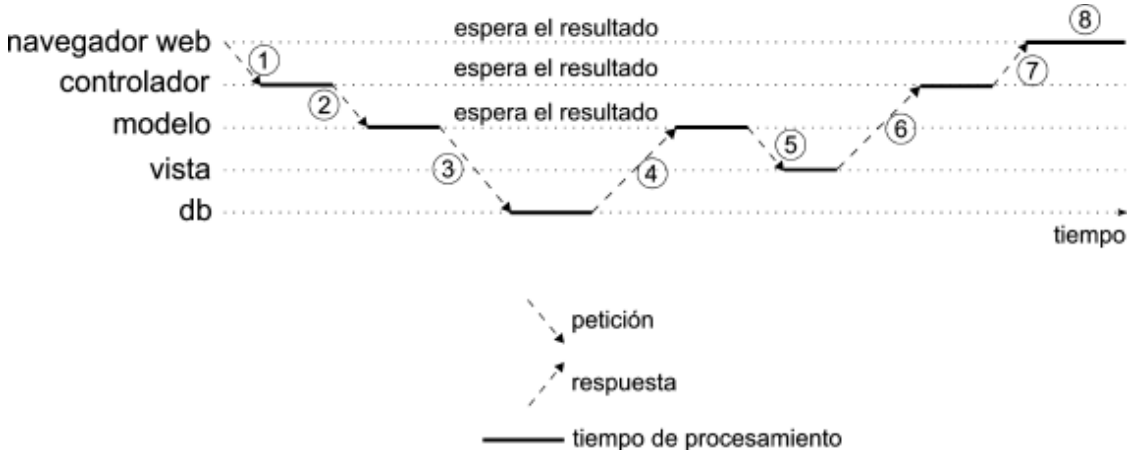


Figura 1.5. MVC – Línea de tiempo del flujo MVC actúa en todas las capas

Como secuencia, a continuación, se puede describir el flujo de MVC de la Figura 1.5, como sigue:

1. El usuario solicita un recurso a través del controlador.
2. El controlador delega el trabajo de traer la información al modelo.
3. El modelo interactuará con la base de datos para traer la información solicitada y dicha información será devuelta al controlador.
4. El controlador delegará la tarea de procesar visualmente los datos en el formato adecuado a la vista.

5. La vista se encargará de generar el resultado visual amigable para el usuario. Renderizará una serie de plantillas que contienen una estructura base del resultado final.

   Este resultado será devuelto al controlador para que realiza las acciones correspondientes.

6. El controlador realizará trabajos para establecer qué tipo de información se está enviando.

7. Posteriormente el resultado generado en el controlador será enviado al usuario quien solicitó la información.

8. Finalmente, el navegador procesará dicha información para presentarla al usuario.

1.3.7. Modularización

La modularización es la manera de administrar la complejidad de un problema mediante la separación de responsabilidades en diferentes capas. Esta modularización no solo está asociada a una organización de la estructura de archivos del sistema sino también, a una organización lógica, es decir, un flujo de datos coherente enfocado en la funcionalidad principal del sistema.

La importancia de considerar modularizar un sistema se da por el hecho de que el software puede evolucionar y escalar a nuevas funcionalidades, e incluso, es cambiante, por lo tanto, quitar otras funcionalidades también es posible. Si en primera instancia no se considera ningún tipo de modularidad en el sistema, es probable que el desarrollo del software se torne complejo, poco escalable y menos comprensible desde el punto de vista de flujo de datos [16], [17].

a. La modularización basada en el patrón de diseño MVC

La organización de archivos de un sistema que utiliza MVC por lo general tiene la estructura mostrada en la Figura 1.6. Esta estructura generalmente ya viene pre-configurada en la mayoría de *frameworks* web.

El código que representa a la vista, va en la carpeta `views`, el código que procesa los datos va en la carpeta `models` y la lógica de manipulación de solicitudes va en la carpeta `controllers` [18]. Esto, por supuesto, va acorde con la definición teórica de MVC. Pero en la práctica, hay otras carpetas y archivos que permitirán complementar la modularización de MVC pues, contienen funcionalidades que no encajan en ninguna de las capas de dicho patrón y, en consecuencia, en ninguna carpeta física.

Por ejemplo, un enrutador, o un módulo que permita enviar correos electrónicos, o ejecutar procesos de cifrado o descifrado de información sensible o, incluso, archivos que son distribuidos en cada petición; archivos que generalmente son denominados *assets* o recursos estáticos y que contienen elementos que serán utilizados por el navegador web; para cada una de éstas funcionalidades también se debe modularizar el sistema de tal forma que puedan estar dentro de una carpeta que indique con mayor facilidad que funcionalidad tienen [18].

Esta modularización se ve reflejada en el uso de carpetas adicionales a MVC (Figura 1.6). Entonces, en la carpeta `assets` estarán archivos estáticos del *front-end*. En la carpeta `utils` se agregarán elementos que proveen funcionalidades como las mencionadas anteriormente. La carpeta `vendor` contiene todas las dependencias de terceros. Finalmente, un archivo `router` contendrá todas las rutas de acceso a los controladores.

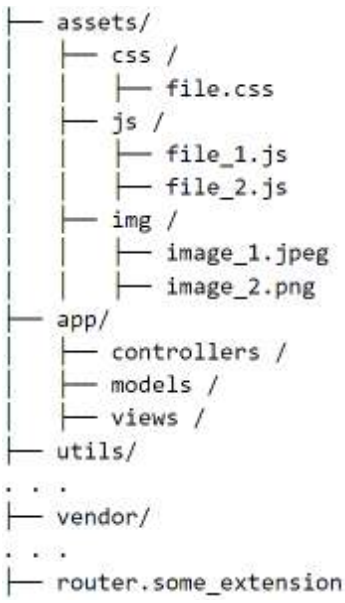


Figura 1.6. Estructura básica de archivos utilizando MVC

Ahora las responsabilidades están distribuidas físicamente de mejor manera. Pero como se ha podido visualizar, el modelo MVC depende de una mayor modularización para que éste pueda ser más eficiente.

Sin embargo, a la estructura MVC mostrada en la Figura 1.6, se le puede dar un enfoque mucho más modular. Esto es; en lugar de centralizar todos los archivos que corresponden a recursos estáticos dentro de la carpeta `assets`, éstos podrían también ser colocados dentro de una carpeta que contenga todos los elementos asociados a una

página. Y solo aquellos que son usados de forma global, podrían ser agregadas dentro de dicha carpeta [19], [20]. Para entender mejor este enfoque se presenta el siguiente escenario.

En una aplicación web, varias páginas pueden tener en común hojas de estilo y *scripting* para varios elementos como la cabecera, el pie de página, un menú, botones, *inputs*, entre otros. Pero, ¿qué sucede con elementos que no tienen los mismos estilos ni tampoco *scripting* Javascript pues no son común entre páginas? Pues, con la estructura MVC de la Figura 1.6, todos, absolutamente todos esos elementos van en un archivo general tanto para los estilos como para el *scripting* Javascript. Sin embargo, se podría dar una modularización a la carpeta `assets` como se muestra en la Figura 1.7.

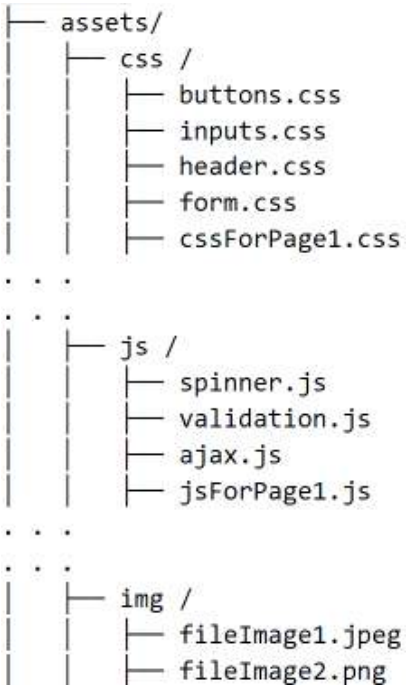


Figura 1.7. Modularización de la carpeta `assets`

Mientras que el manejo de cada elemento se vuelve mucho más amigable, aún es necesario una modularización más profunda puesto que existen elementos que no son comunes, sino que son parte de una página en particular. El siguiente escenario de modularización está basado en una modularización para *frontend* teniendo una estructura basada en páginas y componentes web [21] como se muestra en la Figura 1.8.

Dado que dicha modularización se da a nivel de estructura física de archivos, la definición lógica para cargar todos los recursos, tendría que venir manejado por alguna utilidad la cual permita que sean cargadas todas las dependencias para cada para cada página. Es por eso que se necesita un archivo descriptor el cual contiene todas las dependencias.

Para ello está el archivo `resources.json` que debe contener una estructura base que defina cuales son las dependencias del componente o de la página [19], [20], [21]. La implementación de la aplicación propuesta estará basada en la modularización presentada en la Figura 1.8.

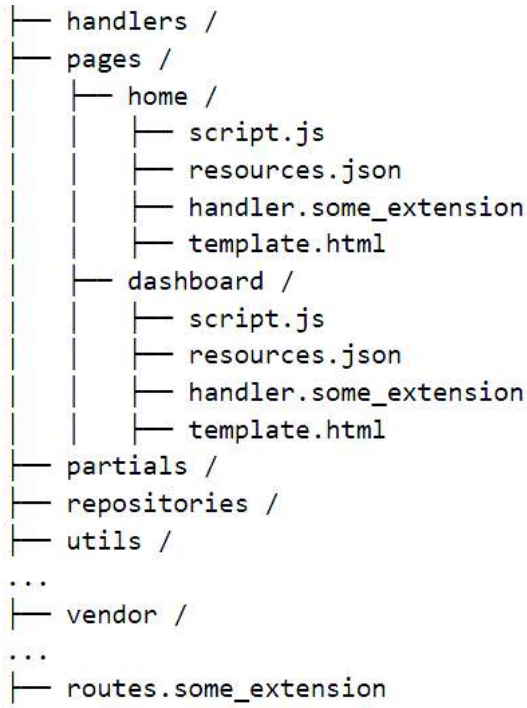


Figura 1.8. Aplicando modularización usando páginas

1.3.8. Contenedores Docker

Docker es una tecnología *open-source* y multiplataforma para generar contenedores. Esto quiere decir que Docker permite empaquetar una aplicación y todas sus dependencias dentro de contenedores Linux. Son utilizados como un mecanismo de "virtualización" ligero a nivel del sistema operativo. Para sistemas operativos como Windows, Docker utiliza otras tecnologías y capas que permiten integrar y soportar los contenedores Linux [22].

La idea fundamental detrás de Docker es contener un ambiente de trabajo completo de tal manera que se tenga una aplicación portable y autosuficiente y que pueda funcionar de la manera prevista sin que existan errores o problemas de compatibilidad. Este empaquetamiento se lo realiza a través de una denominada imagen[18] que contendrá una o varias aplicaciones o servicios, con toda la configuración y sus dependencias

[18] Imagen es un paquete ejecutable que incluye: *runtime*, librerías, variables de entorno, archivos de configuración, entre otros, para ejecutar una aplicación.

correspondiente, para ser ejecutadas sobre otros sistemas operativos o incluso, sobre máquinas virtuales [23].

Los recursos que son utilizados por los contenedores son compartidos con el `kernel` y, en consecuencia, también son compartidos con otros contenedores y procesos propios del sistema operativo base. Sin embargo, todos los recursos utilizados por un contenedor, son aislados en su propio espacio de trabajo (usuario y grupo). En la Figura 1.9 se puede ver el *stack* que conforma un contenedor [24].

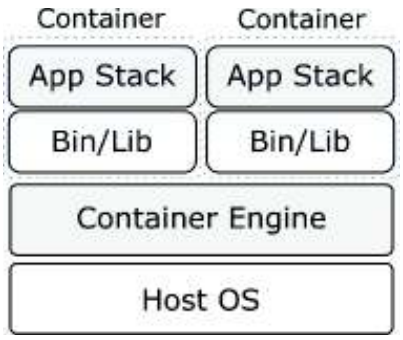


Figura 1.9. Virtualización con tecnología de contenedores

a. Dockerfile

Docker puede construir imágenes de forma automática leyendo un archivo que contiene un conjunto de instrucciones de comandos. Estas instrucciones se encuentran registradas en un archivo con nombre `Dockerfile` que no es más que un archivo de texto plano que contiene *scripts* que indican cuales son las dependencias y configuraciones que formarán parte de la imagen a ser construida. Esta imagen se construye utilizando el comando del Código 1.1. Comando para construir una imagen, ejecutado en la línea de comandos.

```
1    docker build /ruta_alojamiento_imagen
```

Código 1.1. Comando para construir una imagen

El archivo `Dockerfile` contiene no solo comandos propios del *shell* de algún sistema operativo, sino también, instrucciones propias de Docker las cuales permiten realizar ciertas operaciones de forma automática. En el Código 1.2. Códigos más comunes de Docker, se muestran las instrucciones más comunes.

`FROM`: sirve para inicializar una etapa de construcción y establece cual será la imagen base a ser utilizada. Esta instrucción debe ser la primera en ser declarada pues Docker buscará, ya sea en local o en remoto, la imagen base para construir una nueva imagen.

`ADD`: permite copiar archivos locales o remotos, directorios a una ubicación en el sistema de archivos de la imagen.

```
1   FROM       <imagen>
2   FROM       <imagen>:<etiqueta_de_version>
3   ADD        <origen> <destino>
4   ARG        <nombre_variable>=<valor_variable>
5   ENV        <nombre_variable>=<valor_variable>
6   EXPOSE     numero_de_puerto/protocolo
7   WORKDIR    /ruta/de/trabajo
8   RUN        ["executable","param_1",...,"param_n"]
9   CMD        [   "command","param_1",...,"param_n"]
10  ENTRYPOINT ["executable","param_1",...,"param_n"]
```

Código 1.2. Códigos más comunes de Docker

`ARG`: es utilizado para definir variables de entorno temporales mientras se construye la imagen. Su alcance o ámbito está definido únicamente para cada imagen temporal en el proceso de construcción de la imagen final.

`ENV`: a diferencia de la instrucción `ARG`, `ENV` sirve para definir variables de entorno finales las cuales serán utilizadas ya en la ejecución de contenedores.

`EXPOSE`: esta instrucción permite indicar a Docker que el contenedor escuchará en un puerto en particular en tiempo de ejecución.

`WORKDIR`: Esta instrucción permite definir cuál será la ruta actual de trabajo no solo en el proceso de construcción de la imagen, sino también en la ejecución de un contenedor. Si la ruta definida no existe, Docker creará dicha ruta para continuar con el proceso de construcción de la imagen.

`RUN`: esta instrucción permite ejecutar un comando o conjunto de comandos para construir la imagen. Cada instrucción `RUN` que se agregue al archivo `Dockerfile`, en el ciclo de construcción de la imagen, Docker creará una imagen temporal para procesar los comandos en cuestión y, si la ejecución fue exitosa, entonces esos cambios serán agregados a la imagen final.

`CMD`: utilizado para ejecutar comandos para la construcción de una imagen, a través de la línea de comandos.

`ENTRYPOINT`: permite configurar un *script* que será ejecutado cuando una instancia de una imagen sea creada.

b. Construcción de una imagen y ejecución de contenedores

Antes de poder crear un contenedor es necesario crear una imagen. Para obtener dicha imagen, es necesario agregar todas las instrucciones en el archivo `Dockerfile` con todas aquellas instrucciones que permitirán construir la imagen.

Posteriormente de tener generado el conjunto de instrucciones en el archivo `Dockerfile` y, ubicado en el mismo directorio donde se encuentra el archivo, a través de la línea de comandos, se ejecuta el comando presentado en el Código 1.1. Comando para construir una imagen, para construir una imagen.

Docker descargará, instalará y realizará todas las configuraciones de todas las dependencias que hayan sido especificadas en el archivo `Dockerfile`. Este ciclo se puede observar en la Figura 1.10.

Finalmente, luego de obtener dicha imagen, se podrán crear contenedores que no son más que instancias de una imagen, pudiéndose crear *n* instancias de la misma imagen. Esto depende de la capacidad del *host* donde se encuentra ejecutando dicha imagen. Para ejecutar un contenedor se utiliza el comando `docker run nombre_de_la_imagen`. Cada uno de los contenedores tiene su independencia y pueden ejecutarse sobre dichos contenedores comandos como `start`, `restart` y `stop` [25].

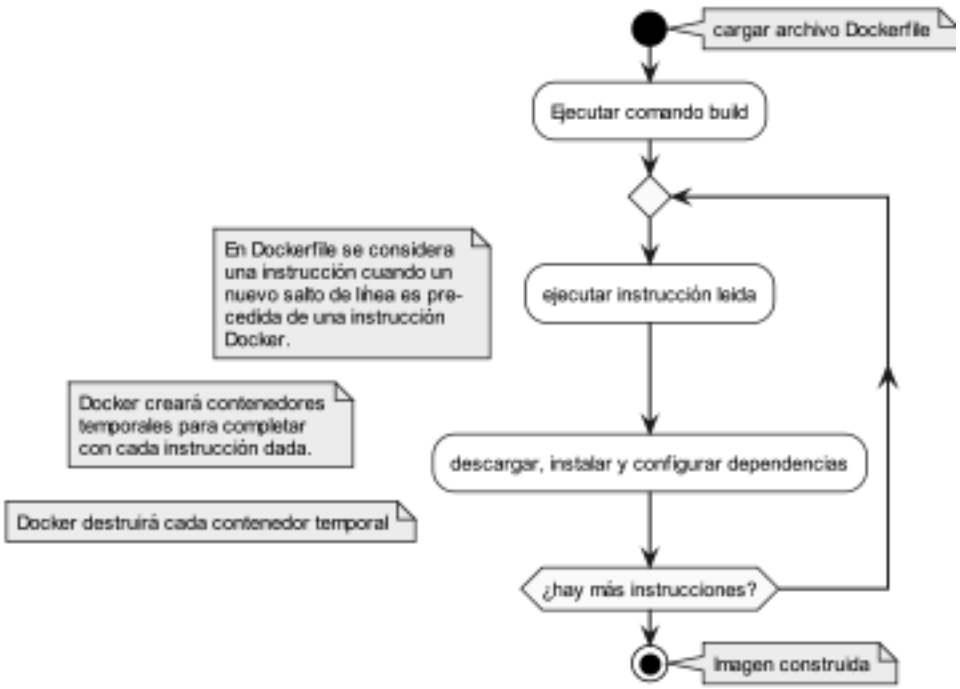


Figura 1.10. Ciclo de construcción de una imagen Docker

1.3.9. WebSocket

WebSocket es un protocolo estándar desarrollado por la IETF (*Internet Engineering Task Force*) y también es una API desarrollada por la W3C (*World Wide Web Consortium*). *Websocket* forma parte de la especificación HTML5 para tecnologías web. *WebSocket* no es más que un socket TCP[19] (*Transmission Control Protocol*) que trabaja sobre la web y permite una comunicación persistente del tipo *full-duplex* y bidireccional entre el cliente web y el servidor web. Esto quiere decir que, tanto el cliente como el servidor, pueden intercambiar información al mismo tiempo en ambos sentidos. Sin embargo, es el cliente quién inicia la petición y el servidor es quién responde a dicha solicitud. *Websocket* es utilizado para construir aplicaciones escalables o en tiempo real [26].

1.3.10. HyperText Transfer Protocol

a. Cookies en HTTP

Según [11], las *cookies* son un conjunto de datos enviados desde el servidor hacia el navegador web del usuario quien realiza una solicitud. Estos datos son almacenados en el navegador con el propósito de ser enviados hacia el servidor cuando un usuario realiza una nueva solicitud. Generalmente las *cookies* son utilizadas para contener información del usuario de tal manera que permita mantener una sesión abierta puesto que el protocolo HTTP no mantiene estados de ningún tipo.

b. Caché en HTTP

En aplicaciones web, generalmente hay cierta información que no cambia con el tiempo. Por ejemplo, hojas de estilo CSS, *scripts* Javascript, imágenes, documentos, etc. Estos recursos son enviados por primera vez desde el servidor hacia el cliente, cuando un usuario realiza una solicitud, y son alojados en carpetas temporales del usuario para posteriormente ser procesadas por el navegador web. Dichos recursos, dado que no cambian, deberían ser reutilizados y no ser solicitados nuevamente cada vez que se realice una petición al servidor.

El caché en HTTP permite que en cada respuesta se establezcan ciertos parámetros de tiempo que indican que el recurso enviado tiene un tiempo de vida útil y, por lo tanto, puede ser reutilizado en lugar de nuevamente ser solicitado al servidor. Esto contribuye a que el número de peticiones realizadas al servidor también disminuya e incluso la carga

[19] TCP es un protocolo fundamental de la red el cual permite conexiones entre dos entidades para transmitir flujo de datos.

de la página se mucho más rápida puesto que los recursos ya no tienen que volver a ser descargados sino más bien reutilizados pues están almacenados de forma local [27].

c. Categorías de códigos de estado de respuesta

A continuación se presentan las categorías de códigos y aquellos códigos de estado de respuesta para HTTP más importantes [28].

1xx - Información: En esta categoría se encuentran aquellos códigos de propósito informativo de algunas acciones que se van a realizar.

2xx - Éxito: Estos códigos indican el estado de éxito de una respuesta ante una solicitud realizada.

3xx - Redirecciones: En esta categoría se encuentran aquellos códigos que indican que se deberá realizar acciones adicionales para poder completar la respuesta ante una solicitud realizará por el cliente.

4xx – Errores del cliente: Indican múltiples estados de respuesta correspondientes a accesos no autorizados, peticiones incorrectas o recursos no encontrados, entre otros.

5xx – Errores del servidor: Indican que se ha producido algún tipo de error, comportamientos no previstos o funciones no soportadas por parte del servidor.

Algunos códigos más comunes son:

Código de estado 200 – OK: Indica que una petición realizada por el cliente se ha ejecutado de manera correcta en el servidor y, en consecuencia, serán enviados al cliente.

Código de estado 201 – Creado: Indica que una petición realizada por el cliente para la creación de un recurso, se ha realizado de forma exitosa.

Código de estado 301 – Movido permanentemente: Este código indica al usuario que el recurso al cual desea acceder, ya no está bajo la misma dirección y que ha sido movido de manera permanente a otra dirección.

Dado que en el resultado se incluye dónde está el contenido actual, el navegador podrá realizar la redirección correspondiente para acceder al recurso solicitado.

Código de estado 302 – Movido temporalmente: Este código indica que el recurso solicitado ha sido reubicado de forma temporal en otra dirección. De igual manera que en el código 301, en el resultado se incluye información de donde se encuentra localizado dicho recurso, por lo tanto, el navegador podrá realizar la redirección correspondiente.

Código de estado 400 – Petición errónea: Este código indica al usuario que la petición realizada contiene un formato no adecuado o que en dicho formato de petición se encuentre alguna variable descrita de forma errónea.

Código de estado 403 – Prohibido: Este código indica al usuario quién realizó la petición, que no puede acceder a dicho recurso pues está protegido y dicho usuario no tiene la autorización correspondiente para poder acceder a ese recurso.

Código de estado 404 – No encontrado: Indica al usuario quien realizó la petición de acceder a un recurso, que dicho recurso no ha sido encontrado ya sea porque ha sido movido a otra dirección, o simplemente ha sido borrado. Sin embargo, esta información no se especifica en el resultado enviado al usuario.

Código de estado 500 – Error interno del servidor: Es utilizado para indicar que se ha producido un error inesperado en el servidor. La información detallada de las causas que generaron este problema no es enviada en la respuesta al cliente.

d. Métodos HTTP para RESTful

Los métodos que a continuación es describen, tienen un enfoque de respuesta que depende del contexto en el que estén trabajando. Particularmente, la descripción de dichos métodos estará enfocado en su uso en RESTful, que es la arquitectura que se maneja para el desarrollo del prototipo [29].

GET: Este método se usa para leer (recuperar) una representación de un recurso. El recurso a ser recuperado se especifica de forma explícita en la URL. Si existe el recurso, GET devolverá una representación en JSON y un código de respuesta 200. Si dicho recurso no existe, se obtiene un error en particular, pero, de igual manera, GET devolverá una representación en JSON y un código de respuesta 404, o 400, según corresponda. Sin embargo, podría haber casos en que el servidor retorne un estado de respuesta 500 cuando haya sucedido algún error en el servidor.

POST: Con frecuencia se utiliza para crear nuevos recursos o enviar información sensible a través de formularios. Sin embargo, POST no es seguro como tal, sino que únicamente la información se la envía en el cuerpo del mensaje HTTP y no en la URL como lo hace el método GET. Cuando la creación de dicho recurso es exitosa, el código de respuesta devuelto será 201. Mientras que, si la operación no se ha podido llevar a cabo, el código de respuesta enviado podría ser cualquiera de la categoría 4xx o 5xx.

PUT: En la mayoría de los casos, este método se utiliza para actualizar de forma parcial o total la información de un recurso existente en el servidor. Sin embargo, PUT también

puede ser utilizado como un mecanismo para crear un recurso, al igual que POST. La diferencia radica en que con el método PUT, el identificador del nuevo recurso se asignará de forma manual indicada en la petición.

DELETE: El nombre asignado prácticamente describe la funcionalidad de dicho método, pues es bastante descriptivo. Este método es utilizado para borrar un recurso el cual debe ser especificado de forma explícita a través de la URL.

1.3.11. Pruebas unitarias

En el ámbito de desarrollo de software, uno de los mecanismos utilizados para evaluar código, es utilizando las denominadas pruebas unitarias las cuales permiten evaluar métodos que tienen la responsabilidad de llevar a cabo una tarea en particular. Estas pruebas son realizadas utilizando librerías especializadas que permiten crear diferentes conjuntos de escenarios de evaluación de tal manera que, al pasar valores de entrada a un método de una clase dada, se pueda comprobar que dicho método, presenta un comportamiento deseado [48], [49], [57].

Para el presente proyecto, se ha considerado utilizar pruebas unitarias como el mecanismo de evaluación del código que es enviado por los participantes. Además, teniendo en cuenta de que el prototipo soporta los lenguajes de programación Java y C# para los torneos de programación, las librerías que serán utilizadas para llevar a cabo las pruebas unitarias son JUnit para Java y NUnit para C#. En la sección 1.16, se muestra el uso de estas librerías. Por otro lado, el prototipo se ejecuta sobre el sistema operativo Linux, por lo tanto, será necesario utilizar compiladores/intérpretes para cada lenguaje que se ejecutan sobre dicho sistema. Estos son el JDK[20] (*Java Development Kit*) para el lenguaje de programación Java y para C#, se utiliza el *runtime* Mono[21] que provee todo lo necesario para que se compile y ejecuten clases de C# sobre algunas distribuciones de Linux.

1.3.12. Lenguajes de programación, maquetado y *scripting*

a. CSharp o C#

C# es un lenguaje desarrollado por el equipo de Microsoft. Es de tipo compilado con un fuerte enfoque en el paradigma orientado a objetos. C# tiene otro tipo de paradigmas

[20] JDK es un conjunto de librerías para compilar e interpretar código fuente de Java [63].
[21] Mono es un proyecto *open-source* compatible con el *framework* .NET de Microsoft el cual permite que código del lenguaje de C# pueda ser compilado, interpretado y ejecutado sobre entornos Linux [64].

asociados, sin embargo, el desarrollo de aplicaciones de software basado en objetos es su foco principal.

Las declaraciones de variables son fuertemente tipadas, es decir, que explícitamente se debe declarar el tipo de dato que va a manejar la variable en cuestión. Sin embargo, C# también ofrece la posibilidad de realizar declaraciones dinámicas basado en la inferencia de tipos[22] y utilizando la palabra clave `var`.

El enfoque que tiene C# para desarrollar aplicaciones es múltiple; no solo permite construir aplicaciones de escritorio sino también aplicaciones web y aplicaciones móviles. Pero gracias al proyecto Mono y la apertura del *core* .NET de Microsoft al mundo *open-source*, se ha logrado que C#, además de otros lenguajes de Microsoft, puedan ser multiplataforma [30].

b. Java

De igual manera que C#, Java es un lenguaje compilado con un fuerte enfoque en el paradigma orientado a objetos. Java se ejecuta sobre una máquina virtual propia de Java. Y a diferencia de otros lenguajes, Java realiza la compilación de su código fuente a un código denominado *bytecode* y que únicamente es interpretado por la máquina virtual de Java (JVM). Java no soporta inferencia de tipos para una asignación dinámica. Java, al igual que C# es multiplataforma y de propósito general [31].

c. Bash – Sh

Antes de definir exactamente que es Bash (*Bourne Again Shell*) y Sh es importante definir lo que es el *shell*.

El *shell* es un intérprete de comandos y un lenguaje de programación de *scripts*. Como un intérprete de comandos, provee una interfaz a las utilidades del sistema operativo GNU/Linux mientras que, como lenguaje, permite definir sentencias de flujo, de control, variables, etc., que serán utilizados para ejecutar dichos comandos [32].

Tanto Bash como Sh son tipos de *shell*. La diferencia radica en que Bash es una actualización de Sh, y contiene más opciones como lenguaje de *scripting* para interpretar comandos [33].

La ejecución de dichos comandos se puede llevar a cabo a través de una consola, también llamado terminal. Sin embargo, cuando se desea automatizar procesos, es decir,

[22] La inferencia de tipos quiere decir que será el compilador quién determine el tipo de variable que se asignará en la declaración inicial.

ejecutar *scripts* de forma automática, la mejor opción es utilizar un archivo de texto plano que contenga todo el *script* que se desea ejecutar con la consideración de que, en dicho archivo, como cabecera principal se agregue la ruta del *shell* que se desea utilizar. Generalmente es `#!/bin/bash` la línea que debe agregarse en cada archivo de *scripting*.

Es común que a dicho archivo se le agrega la extensión `.sh`, sin embargo, no hace falta agregar dicha extensión sino más bien, se deben dar los permisos de ejecución al archivo en cuestión utilizando el comando `chmod +x nombre_del_archivo`, obviamente teniendo agregada la línea correspondiente que indica cuál es el *shell* que se utilizará para interpretar y ejecutar los comandos.

d. Golang

Go, también conocido como Golang, fue iniciado en Google por Robert Griesemer, Rob Pike y Ken Thompson en 2007. La razón principal detrás de este lenguaje fue ayudar a mantener el desarrollo productivo enfocado en la simplicidad y altas prestaciones para aplicaciones de red escalable.

Se ha convertido en el lenguaje de preferencia para los proveedores de infraestructura, y algunos de los proyectos más populares como Docker, entre otros, han sido desarrollados completamente en Go [34].

Go es un lenguaje compilado que soporta declaraciones de tipo estáticas (fuertemente tipadas) y dinámicas (débilmente tipadas). Además, soporta varios paradigmas de programación entre los que más destacan es su capacidad de manejar un modelo de programación concurrente[23] para construir sistemas distribuidos escalables y confiables.

También soporta un modelo de programación orientada a objetos básico pero no hay jerarquía de clases, sino que provee interfaces para proveer funciones (también llamados métodos). Un objeto en Go es llamado `struct` el cual puede contener variables que definen a dicho objeto. También, soporta el paradigma de programación imperativo[24]. Algunos de los puntos clave que hacen de Go un lenguaje popular [35] son los siguientes:

- **Binarios:** Go genera binarios de la aplicación con todas sus dependencias para múltiples arquitecturas de sistemas operativos y procesadores. Esto elimina la

[23] El paradigma de programación concurrente es un mecanismo que permite que varias operaciones puedan ser ejecutadas de forma simultánea en periodos de tiempo superpuestos.
[24] El paradigma de programación imperativo describe explícitamente una secuencia de pasos bien definidos que son dados a la computadora para que realice una tarea en particular y así obtener el resultado deseado.

necesidad de que se instale el *runtime* de Go para ejecutar la aplicación construida.

- **Diseño simplificado:** los creadores de Go tomaron la decisión de mantener un lenguaje simple y fácil de entender. Se prefiere la composición[25] en lugar de la herencia[26] y tiene un sistema de tipos amigable que no solo soporta declaraciones fuertemente tipadas sino también débilmente tipadas basado en la inferencia de tipos.

- **Soporte de concurrencia:** Este es uno de los puntos clave donde destaca Go de otros lenguajes pues propone un nuevo mecanismo que permite manejar la simultaneidad. Para ello, soporta funciones de primera clase las cuales permitirán trabajar con canales y las llamadas Goroutines. La simultaneidad le permite utilizar eficientemente la capacidad del procesador, incluso si este solo tiene un núcleo. Esto le permite tener cientos de miles de Goroutines concurrentes (hilos ligeros) ejecutándose en un solo procesador.

Para garantizar la eficiencia y escalabilidad, el compilador de Go tiene ciertas consideraciones en el instante de realizar la compilación del código fuente. Una de las más importantes es que el compilador de Go verifica, en tiempo de compilación, que todas las declaraciones sean utilizadas. Si no es así, se mostrará un error en tiempo de compilación y se acabará la ejecución del mismo y no compilará hasta que dicha declaración sea quitada. También existe una verificación de dependencias circulares. Para evitar estos problemas de circularidad, el compilador de Go mostrará un error en tiempo de compilación de dicho problema, terminando la ejecución del mismo hasta que se elimine la circularidad [34].

d.1 Dependencias en Go

Después de la cláusula `package`, cada archivo fuente puede tener uno o más enunciados de importación de dependencias. La palabra clave utilizada para ello es `import` y, a continuación, se escribe en forma de lista, una cadena de texto que identifican cada una de las dependencias, como se muestra en las líneas de código 3 a la 6 del Código 1.3.

[25] La composición es un mecanismo de reutilizar código a través de instancias de clases. Varias funciones pueden delegar una tarea en particular a otra función que sabe cómo hacer dicha tarea.
[26] La herencia es el mecanismo que permite que un objeto pueda heredar todas las propiedades y métodos de otro objeto. La herencia es uno de los pilares fundamentales de la programación orientada a objetos.

```
1  import nombre_del_paquete
2  import (
3      "nombre_de_la_dependencia_1"
4      "nombre_de_la_dependencia_2"
5      ...
6      "nombre_de_la_dependencia_n"
7  )
```

Código 1.3. Importar un paquete en Golang

d.2 Declaración de variables

Se definen dos mecanismos para declarar variables. La primera es utilizando la palabra clave `var` que generalmente se utiliza como una declaración de tipo global. O la otra forma es utilizando los signos clave `:=` en el que, al lado izquierdo, va el nombre de la variable y al lado derecho, el valor correspondiente sin especificar, en ninguno de los dos lados, el tipo de variable que está asociada. En este caso es una declaración de variable de tipo local y su ámbito está dado únicamente por la función que la contiene. En el Código 1.4 se presentan algunos ejemplos.

```
1  var a, b int
2  var (
3      c int
4      d string
5      e bool
6  )
7  f, g, h := 10, "texto", true
```

Código 1.4. Formas de declaración de variables en Golang

En la línea 1 del Código 1.4 se muestra la declaración de dos variables (`a` y `b`) del mismo tipo y están separadas por coma, además, son de tipo global pues contiene la palabra clave `var` al inicio de la declaración. A diferencia de otros lenguajes de programación, la asignación de tipo, se lo realiza indicando el tipo de dato, al final de la declaración de las variables.

Si se requiere asignar varios tipos de datos en un ámbito global, se debe encerrar entre paréntesis la declaración de variables, seguidas por el tipo de dato como se ve en las líneas de código de la 2 a la 6 en el Código 1.4. En la línea 7, se puede ver que se declaran variables separándolas por coma y, a continuación, después del signo `:=`, la asignación de valores.

El primer valor se asigna a la variable de tipo entero llamada `f`. El segundo valor de tipo `string`, se asigna a la variable `g` y, finalmente, el tercer valor de tipo `bool`, se asigna a

la tercera variable con nombre `h`. En este tipo de declaración, las variables son de tipo local y, además, Go determinará, a través de la inferencia de tipos, el tipo de variable que corresponde a cada variable declarada (`f`, `g` y `h`).

d.3 Declaración y visibilidad de funciones

La palabra reservada para declarar funciones es `func` y a continuación, el nombre de la función. La consideración es que, para la visibilidad de funciones, sean estas públicas o privadas, se deben seguir algunas especificaciones de capitalización. Para funciones de tipo pública se debe tener en consideración el tipo de capitalización *upper case*, y *lower case*, para funciones de tipo privadas.

d.4 Goroutines

Como especificación del lenguaje, las *Goroutines* son como una función típica de Go, sin embargo, dichas funciones, pueden ser ejecutadas de manera simultánea con otras, independientemente si son o no *Goroutines*.

Las *Goroutines* vienen a ser subprocesos del sistema operativo que son atendidos en tiempo de ejecución. Estas son consideradas como hilos ligeros, pues para crear dichas *Goroutines*, no se requiere de altos consumos de memoria ni de procesamiento en comparación a un hilo del sistema como en otros lenguajes de programación.

Este bajo consumo de recursos permite que Go pueda tener miles de *Goroutines* ejecutándose de forma simultánea. Además, Go se encarga de abstraer todos los detalles intrínsecos de manejar las *Goroutines* y entrega una API amigable para trabajar con concurrencia.

Se utiliza la palabra reservada `go` antes de la llamada a una función, para inicializar una Goroutine. En el Código 1.5, se muestra un ejemplo del uso de las *Goroutines*.

```
1   package main
2   import (
3       "fmt"
4   )
5   func ejemploGoroutine() {
6       fmt.Println("ejecución de una Goroutine")
7   }
8   func main() {
9       go ejemploGoroutine()
10      fmt.Println("función principal")
11  }
```

Código 1.5. Uso de Goroutines

En la línea número 9 del Código 1.5, la palabra reservada `go` indica que la función `ejemploGoroutine` será una *Goroutine*.

Ahora, la función `main` y `ejemploGoroutine` se ejecutarán de forma simultánea. Sin embargo, cada una de ellas estará en su propio subproceso.

Si se ejecuta el ejemplo del Código 1.5, el programa solo mostrará el texto "`función principal`", pero no se muestra el mensaje de la función `ejemploGoroutine`.

Para entender el comportamiento de ejecución de *Goroutines* es importante mencionar dos aspectos fundamentales.

1. Cuando una *Goroutine* se inicia, el control de la ejecución de llamada se devuelve al programa de forma inmediata para poder ejecutar las sentencias que se encuentran a continuación de la *Goroutine* y se ignoran los valores de devolución de la *Goroutine*.
2. La *Goroutine* principal debería estar funcionando para que se ejecute cualquier otra sentencia o, incluso, cualquier otra *Goroutine* secundaria.

Dadas las consideraciones anteriormente mencionadas, la ejecución de la *Goroutine*, en el Código 1.5, no bloquea la ejecución de la siguiente instrucción puesto que el control fue retornado de forma inmediata a la *Goroutine* principal sin esperar el valor de resultado de la *Goroutine*, generándose que la *Goroutine* principal, termine su ejecución antes de que la *Goroutine* secundaría pudiera haberse ejecutado y, es por eso que no se muestra el mensaje.

Para ver el resultado de las funciones `main` y `ejemploGoroutine` en consola, se puede modificar el Código 1.5 de tal manera que la *Goroutine* principal no termina su ejecución de forma inmediata, y la *Goroutine* secundaría tenga tiempo de ejecutarse, como se muestra en el Código 1.6.

Al ejecutar el Código 1.6, el resultado será que, en primera instancia, se muestra el mensaje "`ejecución de una Goroutine`" y, posteriormente, el mensaje de "`función principal`". Esto es gracias a la línea 10 la cual permite que la *Goroutine* principal no termine su ejecución después de iniciada la *Goroutine* secundaria sino más bien, haya un lapso de un segundo para ejecutar la siguiente línea de código.

En este proceso, la *Goroutine* secundaría tiene el tiempo suficiente para poder ejecutarse, antes de que la *Goroutine* principal termine su ejecución.

Sin embargo, en un contexto real, este mecanismo de dar un lapso de tiempo para bloquear la *Goroutine* principal no es el adecuado ni tampoco recomendado sino más bien ha sido utilizado para propósitos demostrativos de la ejecución de las *Goroutines* secundarias.

```
1   package main
2   import (
3       "fmt"
4   )
5   func ejemploGoroutine() {
6       fmt.Println("ejecución de una Goroutine")
7   }
8   func main() {
9       go ejemploGoroutine()
10      time.Sleep(1 * time.Second)
11      fmt.Println("función principal")
12  }
```

Código 1.6. `TimeSleep` y Goroutines

Para contextos reales, los canales se pueden usar para bloquear la *Goroutine* principal hasta que las *Goroutines* secundarias terminen su ejecución.

d.5 Canales

Uno de los aspectos más importantes en la concurrencia es que las *Goroutines* puedan comunicarse entre sí. Para ello, Go implementa los denominados canales que no son más que tuberías que permiten que las *Goroutines* puedan tener comunicación entre ellas. Algo que hay que destacar de los canales es que, cuando se trabaja conjuntamente con las *Goroutines*, dichos canales evitan que se produzcan las condiciones de carrera[27] al acceder a la memoria compartida.

La palabra reservada para declarar canales en Go es `chan`. Para crear un nuevo canal se debe utilizar la palabra reservada `make`. Cada canal creado tiene asociado un tipo de dato que el canal puede transportar. Una forma de crear un canal se muestra en el Código 1.7.

```
1   a := make(chan tipo_de_dato)
```

Código 1.7. Declaración de canal en Golang

[27] Las condiciones de carrera ocurren cuando dos o más procesos pueden acceder a recursos compartidos y estos intentan cambiar dichos recursos al mismo tiempo.

La sintaxis de declaración para definir variables que envían y reciben información a través de un canal se presenta en el Código 1.8.

```
1    canalA := make(chan tipoDeDato)
2    datosDesdeElCanal := <- canalA
3    a <- datosHaciaElCanal
```

Código 1.8. Formas de recibir y enviar información a través de un canal

En la línea 1, se realiza la declaración e inicialización de un canal. En la línea 2 se lee el valor leído del `canalA` y se almacena en la variable `datosDesdeElCanal`. En la línea 3 se envían datos al canal.

En el Código 1.9 se presenta un ejemplo de canales en lugar del uso de un `time.Sleep` para bloquear la ejecución de la *Goroutine* principal hasta que termine la ejecución de la Goroutine secundaria. En la línea 10, se declara e inicializa un canal de tipo `bool`. En la línea 11, se inicia una *Goroutine* pasándole como parámetro el canal creado anteriormente.

En la línea 12, se espera a que la *Goroutine* escriba algún valor en el canal dado y mientras no se realice esta acción, esta línea estará bloqueando el control de la *Goroutine* principal y, por lo tanto, no se ejecutará la siguiente línea hasta que se haya completado la operación anterior. Cuando se reciben datos en el canal, no se almacena en ninguna variable el resultado. A continuación, se desbloquea y retorna el control a la *Goroutine* principal y, por lo tanto, puede ejecutar la línea 13 para finalmente, terminar la ejecución del código.

```
1    package main
2    import (
3        "fmt"
4    )
5    func ejemploGoroutine(hecho chan bool) {
6        fmt.Println("ejecución de una Goroutine")
7        hecho <- true
8    }
9    func main() {
10       hecho := make(chan bool)
11       go ejemploGoroutine(hecho)
12       <- hecho
13       fmt.Println("función principal")
14   }
```

Código 1.9. Ejemplo de uso de Goroutines y canales

d.6 Concurrencia en Go

De forma general, la concurrencia es un mecanismo el cual permite que varias operaciones puedan ser ejecutadas en periodos de tiempo superpuestos, teniendo como resultado, una ejecución de operaciones como si estas estuvieran ejecutándose al mismo tiempo.

Este proceso de concurrencia se da a nivel de programación pues a nivel de máquina, todas las operaciones son ejecutadas de forma secuencial, es decir, una operación tras otra, ordenadas a través del tiempo.

La ventaja de este modelo de programación se ve reflejado cuándo existen un sinnúmero de ejecuciones que necesitan ser llevadas a cabo, casi de forma simultánea.

En Go, el modelo de concurrencia es parte inherente del lenguaje y se da usando *Goroutines* y canales. Este modelo en Golang, difiere de la concurrencia en otros lenguajes de programación pues como tal, esos lenguajes utilizan los denominados hilos del sistema los cuales llegan a consumir varios megabytes de memoria, mientras que la concurrencia en Golang se da a través de los canales y *Goroutines*.

Estos últimos llegan a consumir solo unos cuantos kilobytes de memoria. Por lo tanto, lo hace candidato perfecto para manejar aplicaciones realmente escalables por su bajo consumo de memoria cuando de concurrencia se trata [36].

d.7 Servidor web en Golang

Otra de las características que también destacan en Go es su manera fácil de implementar un servidor web. Esto es gracias a las librerías estándar base que ayudan a desarrollar de forma rápida aplicaciones o servicios web.

En el Código 1.10 se muestra cómo realizar la implementación de un servidor web básico con una URL principal de acceso. Para ello, como se ve en la línea 2, se importa la librería `net/http` pues esta contiene todas las funcionalidades que permiten implementar el servidor web.

En la línea 9, se utiliza el método `HandleFunc`, el cual recibe como primer parámetro una URL que será un punto de acceso al servidor web, y como segundo parámetro, una función que manejará los detalles de la petición y generará la respuesta que se solicita (líneas 3 a 7). Después, se debe especificar un puerto y otros parámetros de configuración en el método `ListenAndServe`, para que este, se encargue de todos los detalles de ejecución y manejo de peticiones (línea 10).

Para ejecutar el servidor del Código 1.10, desde la línea de comandos se ejecuta el comando `go run nombre_archivo.go`. Con ello, Golang pondrá en funcionamiento el servidor web y, a través de un navegador web, se podrá acceder al sistema mediante la URL que ha sido especificada, en este caso es la ruta principal, por lo tanto, se accede vía `http://localhost:8080`, y se obtendrá como respuesta en el explorador web "`página principal`".

```
1   package main
2   import "net/http"
3   func paginaPrincipal(
4       w http.ResponseWriter,
5       r *http.Request) {
6       w.Write([]byte("página principal"))
7   }
8   func main() {
9       http.HandleFunc("/", paginaPrincipal)
10      err := http.ListenAndServe(":8080", nil)
11      if err ¡= nil {
12          panic(err)
13      }
14  }
```

Código 1.10. Código básico de un servidor web en Golang

d.8 Servidor de contenido estático en Golang

Un contenido estático es aquel cuyo contenido no cambia con el tiempo. Por lo general, este tipo de contenidos tiene que ver con imágenes, plantillas HTML base, *scripts*, hojas de estilo, entre otros.

Para distribuir este tipo de contenidos se puede hacer uso de alguna librería sobre un servidor web, la cual permitirá definir en la cabecera de respuesta HTTP, que tipo de contenido se está enviado desde el servidor hacia el cliente, para que el cliente pueda interpretar dicho contenido y, en consecuencia, pueda procesar la información provista desde el servidor.

Si no se especifica en la cabecera de respuesta HTTP que tipo de contenido está siendo enviado desde el servidor, el navegador web no sabrá como manipular la información y solo presentará la información en texto plano [37].

Go maneja en el paquete `net/http`, todos los detalles de un servidor web incluido las funcionalidades para distribuir contenidos de tipo estático. Para que el servidor pueda distribuir dicho contenido, en el Código 1.10, se debe agregar el Código 1.11, el cual

contiene un manejador para distribuir archivos y, únicamente, se debe indicar en que ruta se encuentran los archivos estáticos. Como se muestra en el código la ruta es `./static`.

```
1    http.Handle("/", http.FileServer(http.Dir("./static")))
```

Código 1.11. Manejador de contenido estático

d.9 Motor de renderizado de plantillas en Golang

Un motor de renderizado de plantillas es un pre-procesador de texto que manipula variables y estructuras de datos como *strings*, *arrays*, etc., para realizar validaciones, bucles, entre otros, con el propósito de generar contenido HTML que será presentada al usuario final.

Esto permite que el servidor pueda proveer contenido dinámico utilizando una plantilla base [38]. Golang dispone de la librería `html/template` la cual provee de un motor de renderizado con *helpers* básicas para transformar estructuras de datos en texto con etiquetas HTML.

A continuación, se muestran brevemente, algunos de los *helpers* que son utilizados para manipular estructuras de datos [39] básicos.

El elemento clave el cual indica que se trata de información que debe ser manipulada va contenida dentro de llaves: `{{.Estructura}}`. Para recorrer estructuras de datos de tipo objeto o *array* se utiliza el *helper* `{{ range $value := .Estructura }}`.

Para realizar comparaciones se utiliza el *helper* `{{if comparacion .Estructura}}`. Donde `comparacion` viene dado por:

- `eq`: utilizado para realizar la comparación de tipo ==
- `ne`: utilizado para realizar comparación del tipo !=
- `lt`: utilizado para realizar una comparación del tipo <
- `le`: utilizado para realizar una comparación del tipo <=
- `gt`: utilizado para realizar una comparación del tipo >
- `ge`: utilizado para realizar una comparación del tipo >=

Para extender la funcionalidad de la librería y proveer nuevos *helpers*, Golang dispone de una estructura del tipo `clave:valor` en el cual, la clave representa el nombre del *helper*

y el valor, una función que ejecutará un código para cumplir con el objetivo deseado. Para realizar un *helper* se puede utilizar el Código 1.12.

e. HyperText Markup Language (HTML)

HTML es un lenguaje de marcado que contiene etiquetas para representar la estructura de una página web. Cada una de estas etiquetas representan un nodo dentro del documento HTML y todas en conjunto permiten generar el DOM del documento utilizando Javascript.

Estas etiquetas pueden simbolizar bloques de texto informativo como párrafos, cabeceras, *inputs*, imágenes, vídeos, recursos externos a través de los llamados *iframes*, entre otros.

```
1   template.Funcs(template.FuncMap{
2       "nuevo_helper": func(dato tipo_de_dato) tipo_de_dato_devuelto {
3           //ejecutar Código para operar sobre el dato a ser procesado
4           return tipo_de_dato
5       }
6   })
```

Código 1.12. Código para extender *helpers* en *templates* de Golang

Con frecuencia, cada etiqueta HTML contiene selectores asociados como atributos que pueden ser utilizados para relacionarse con selectores de CSS o inclusive, como identificadores para generar eventos o capturar información con Javascript.

Además, cuenta con atributos propios de la versión de HTML5 como el atributo `data-*` que puede contener información extra, como se muestra en el Código 1.13.

Cualquier información dentro de atributos HTML, pueden ser capturados mediante Javascript.

```
1   <div id="extra-info" data-info="texto informativo"></div>
```

Código 1.13. Declaración del atributo `data-*` de HTML5

En el código 1.14, se puede visualizar la estructura de código de una página de HTML.

La línea 1, es una etiqueta que indica que se trata de un documento HTML. En las líneas 2 a 7, contienen etiquetas que son utilizadas para definir información adicional de la página como por ejemplo el título u otro tipo de información. Además, dentro de las etiquetas `style` (línea 5), se puede agregar contenido CSS. Dentro de la etiqueta `body` (línea 9 a 14), se agrega toda la información que será presentado al usuario. Además, se

puede agregar código Javascript dentro de la etiqueta `script` (línea 11 a 13) para ejecutar acciones sobre el DOM.

```
1   <html>
2   <head>
3       <!-- cabeceras de información -->
4       <title>Document</title>
5       <style>
6           /*etiquetas CSS*/
7       </style>
8   </head>
9   <body>
10      <!-- etiquetas HTML -->
11      <script>
12          //scripts de javascript
13      </script>
14  </body>
15  </html>
```

Código 1.14. Código de ejemplo HTML

f. Cascade Style Sheet (CSS)

Es un lenguaje de maquetación utilizado para definir el diseño de una página web. CSS permite que el resultado de las páginas sea visualmente mucho más atractivo a diferencia de presentar únicamente texto sin formatos ni colores o animaciones.

El navegador es el encargado de procesar la información contenida en un archivo con extensión CSS y, conjuntamente con la página HTML que envía el servidor, son asociados mediante selectores las etiquetas de HTML y CSS para ser renderizados.

En CSS hay varios tipos de selectores. Estos selectores pueden ser explícitos utilizando el mismo nombre de la etiqueta de HTML o selectores que contienen identificadores que son utilizados de forma global en un documento HTML.

f.1 Selectores de tipo

Estos selectores coinciden con el nombre de una etiqueta HTML. Por ejemplo, en HTML se tienen etiquetas de párrafos declarados únicamente con la letra `p`; etiquetas de cabeceras son identificados con la letra `h` precedida de un número, por ejemplo, `h1`, `h2`, `h3`, `h4`, `h5`, `h6`, entre otros.

Estos mismos nombres se pueden utilizar como declaración en una hoja de estilo, como se muestra en el Código 1.15. Cuando se usa la notación del Código 1.15, absolutamente

todas las etiquetas que coinciden con los identificadores, serán afectadas por el estilo dado en el documento CSS.

```
p {
    font-size: 12px;
}
h1, h2, h3, h4, h5, h6{
    color: red;
}
input {
    width: 100px;
}
```

Código 1.15. Declaración de selectores de tipo en CSS

f.2 Selectores descendentes

Un documento HTML contiene etiquetas que son anidadas en cascada en otras etiquetas. Como se muestra en el Código 1.16.

```
<h1>Título uno</h1>
<div>
    <h1>Título dos</h1>
    <p>Párrafo</p>
</div>
```

Código 1.16. HTML con etiquetas anidadas

Si se utiliza un selector de tipo para dar un estilo a la etiqueta `h1`, como en el ejemplo mostrado en el Código 1.15, CSS aplicaría estilos a todas las etiquetas de tipo `h1`. Para evitar esto se puede definir un selector descendente como se presenta en el Código 1.17.

```
div h1 {
    ...
}
```

Código 1.17. Declaración de selectores descendentes en CSS

f.3 Selectores de clase

Los selectores de clase son útiles para especificar el diseño únicamente a aquellos elementos que contienen un identificador asociada mediante la palabra clave `class`. Un ejemplo de documento HTML que incluye el uso de clases se muestra en el Código 1.18.

En la hoja de estilos, para acceder al selector de clase y aplicar estilos, se debe agregar un punto antes del identificador como se muestra en el Código 1.19.

```html
<h1>Título uno</h1>
<div>
    <h1>Título dos</h1>
    <p class="titulo">Párrafo</p>
</div>
<div>
    <h1>Título tres</h1>
    <p>Párrafo</p>
</div>
```

Código 1.18. HTML con `h1` externo y anidados dentro de la etiqueta `div`

Al utilizar este mecanismo, los estilos se aplicarán únicamente a aquellos elementos que coincidan con el selector de clase indicado en el atributo `class` en la etiqueta HTML.

```css
.titulo {
    color: black;
    font-size: 12px;
    font-weight: bold;
}
```

Código 1.19. Declaración de selector de clase en CSS

f.4 Selectores de identificación única

Cuando se desea agregar estilos a un único elemento HTML, se utiliza el atributo `id` el cual permite que, a través de la hoja de estilo, se apliquen estilos al elemento que coincide con el `id`.

A diferencia del atributo `class` que permite que todos los elementos con el mismo identificador tengan el mismo estilo, con el atributo `id`, si dos elementos tienen el mismo identificador, los estilos solo se aplicarán al último elemento encontrado en el documento HTML.

Para acceder al selector de identificación única se debe agregar el signo `#` antes del identificador, como se muestra en el Código 1.20.

```css
#elemento-unico {
    color: black;
    font-size: 12px;
    font-weight: bold;
}
```

Código 1.20. Declaración de selector de identificación única

g. Javascript

Es un lenguaje de *scripting* débilmente tipado, manejado por eventos, y basado en prototipos para su orientación a objetos.

Es comúnmente utilizado en el lado del cliente a través del navegador web para manipular el DOM de una página web. Esta manipulación incluye no solo alterar el contenido de la página sino también, generar animaciones visualmente agradables. Además, con el avance de las tecnologías, este lenguaje puede ser utilizado en el *backend* como un lenguaje del lado del servidor, o, realizar una interacción con el servidor como un proceso en *background* para traer nueva información sin que la página sea recargada. Esto es gracias a tecnologías como AJAX y *WebSockets*.

Para acceder a cualquier elemento que se encuentra en el DOM de una página web, Javascript provee métodos que permiten capturar los elementos a través de su identificador único o identificador de clase. Un ejemplo de cómo capturar el atributo `data-*` del Código 1.13, utilizando Javascript, se muestra en el Código 1.21.

```
1   <script type="text/javascript">
2       var data = document.getElementById('extra-info')
3   </script>
```

Código 1.21. Captura de atributos `data-*` de HTML5 vía Javascript

Para obtener varios elementos que contienen una misma clase se utiliza el método `getElementByClassName` pasándole como atributo, el nombre de la clase como se muestra en el Código 1.22.

```
1   <script type="text/javascript">
2       var data = document.getElementsByClassName('extra-info')
3   </script>
```

Código 1.22. Captura de atributos `class` de HTML vía Javascript

g.1 Javascript manejado por eventos

Javascript fue desarrollado para ser un lenguaje que trabaja con un único hilo y donde la asincronía es muy importante para generar procesos no bloqueantes, los cuales son manejados por eventos. Este modelo de programación determina el flujo de una secuencia de instrucciones a través de un evento en particular, ya sea generado por una interacción del usuario sobre una parte en particular de la página web a través de clics en elementos del DOM, o la delegación de eventos del mismo programa, como por ejemplo mediante el uso de AJAX.

Para manejar este modelo, se debe tener en claro dos puntos importantes:

1. Un controlador de eventos es una función de devolución de llamada, también conocida como *callback*, que se invocará cuando se desencadene un evento como, por ejemplo, eventos que son invocados por Javascript o, eventos que se ejecutan por acciones que realiza el usuario como `clic`, `mouseover`, `keyup`, `keydown`, entre otros.

2. Un bucle de eventos que escucha por los desencadenadores de eventos, el *callback* asociado al evento correspondiente que será llamado para que sea ejecutado.

g.2 Callback en Javascript

Un *callback* es una función de orden superior[28] que se pasa como un parámetro a otra función. Esta función se invoca justo después de que se haya completado alguna operación. Este proceso se desarrolla de forma asíncrona y en un subproceso único con la ayuda de un bucle de eventos.

En el Código 1.23 se muestra un ejemplo de cómo se pasa un *callback* a una función cualquiera. En las líneas 1 a 8, se declara una función llamada `funcionA` que recibe 3 parámetros que pueden ser de cualquier tipo de datos (primitivos, arreglos, *string*, funciones, etc.). Cuando se llama a `funcionA`, en la línea 10, el tercer parámetro que se pasa se vuelve un *callback* y se agrega al bucle de eventos.

```
1   function funcionA(arg1, arg2, arg3) {
2       /**
3        * realiza otras acciones. Pueden
4        * tomar cualquier tiempo en
5        * ejecutarse.
6        */
7       arg3(a, b)
8   }
9
10  funcionA(dato_1, dato_2, function (p, q) {
11      /*
12       * ejecutar cualquier acción con la
13       * variable resultado
14       */
15  })
```

Código 1.23. Ejemplo de uso de un *callback* en Javascript

[28] Una función de orden superior es aquella función considerada de primera clase o también conocida como ciudadana de primera clase, que no necesita ser instanciada para poder ejecutarse.

g.3 Bucle de eventos en Javascript

Para describir el comportamiento del bucle de eventos, hay que tener en cuenta la Figura 1.11, en el cual hay una serie de elementos que permiten manejar los *callbacks* en Javascript.

El primer elemento es el Javascript *Engine* el cual contiene dos sub-elementos; un *heap* que es una porción de memoria donde dinámicamente se almacenarán las declaraciones de las funciones, y un *stack*[29] de funciones que permite realizar un seguimiento de la función que se está ejecutando actualmente y qué función se va a ejecutar después de que la actual termine su ejecución. Es importante mencionar que el intérprete de Javascript irá agregando primero aquellas funciones que se encuentran al final del *script* hasta llegar a la parte superior [40] .

Cuando se llama a una función que forma parte de la *Webapi*, o se realiza una operación asíncrona, este se ejecutará primero, pero ya no formará parte del *stack* de funciones, y, después de terminada su ejecución, los *callbacks* que forman parte de la función que las contiene, serán enviadas al *callback queue*[30].

Como el *event loop* es un sub-proceso que se está ejecutando constantemente con el propósito de escuchar y verificar el *callback queue*, este irá quitando las funciones de la *queue* y agregándolas al *stack* de funciones para que puedan ser ejecutadas.

Los *callbacks* ayudan a que las acciones en el navegador no queden bloqueadas y el usuario pueda continuar con un flujo normal de navegación sobre la página web. Sin tener en consideración estos aspectos importantes de manejar la asincronía y procesos no bloqueantes, se pueden tener procesos bloqueantes y por lo tanto, impedir que el usuario ejecute alguna acción sobre la página web hasta que el proceso que bloquea toda la aplicación, termine su ejecución [41].

1.3.13. Librerías y frameworks

A continuación, se describen las librerías y *frameworks* más importantes que son utilizados para construir todos los elementos que conforman el prototipo. Se utilizan estas librerías con el propósito de optimizar el trabajo que se realiza en *frontend* para construir las interfaces de usuario realmente atracticas e interactivas.

---

[29] Un *stack* es una estructura de datos donde se agregan elementos en la parte superior y solo se elimina el último elemento que fue ingresado. De otra forma, apilar y desapilar elementos en el *stack*.

[30] Un *queue* es una estructura de datos similar a un *stack*. Los elementos son agregados al final del *queue*, y solo se puede eliminar el primer elemento. De otra forma, encolar y desencolar elementos en la *queue*.

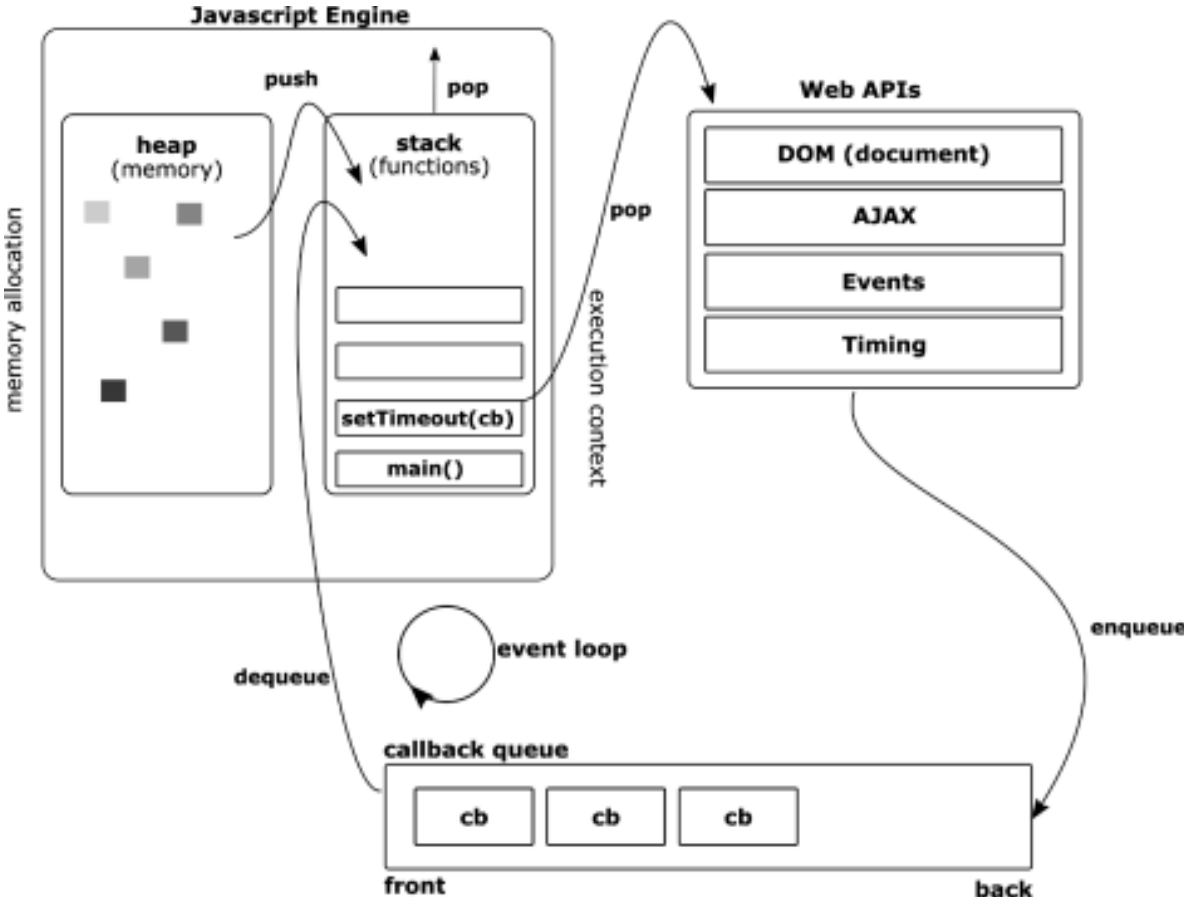


Figura 1.11. Bucle de eventos en Javascript

h. Vue.js

Vue.js también llamado como Vue (pronunciado como *view*), es una *framework* Javascript *open-source* que permite construir interfaces de usuario interactivas y componentes web en cualquier contexto en el que Javascript pueda ejecutarse. Además, es soportado por una gran cantidad de navegadores web.

Vue permite simplificar el desarrollo de aplicaciones web o elementos que pueden ser embebidos en una página web puesto que provee una API realmente potente para modificar el DOM de una página. Vue se centra en la capa vista y puede integrarse de forma sencilla con cualquier otra librería y *framework* de Javascript [42].

Las características más importantes de Vue son [43]:

- **Plantillas:** usa plantillas con una sintaxis de declaración basada en la especificación de HTML.

- **DOM virtual:** Todos los cambios realizados sobre las plantillas, son realizados en un DOM virtual que no es más que una copia del DOM real que representa la estructura de datos. Posteriormente, Vue re-renderiza el DOM real a través de la comparación con el DOM virtual para aplicar los cambios correspondientes solo en aquellos elementos que han sido modificados.
- **Enlace de datos:** ayuda a manipular o asignar valores a cada uno de los atributos de las etiquetas HTML. Esto se lo hace a través de la directiva `v-bind` propia de Vue
- **Manejo de eventos:** los eventos son manejados a través de la directiva `v-on`. En HTML, los eventos pueden representar cualquier acción que se realice sobre la interfaz de usuario o incluso, sobre la página como tal.
- **Directivas:** Vue tiene directivas que son agregadas ya sea como atributos de cada etiqueta HTML o como una etiqueta que contiene etiquetas HTML. Éstas son utilizadas para realizar múltiples acciones sobre la interfaz.
- **Escuchadores:** son elementos que estarán escuchando todos los cambios que se generen en cada una de las etiquetas HTML.

A continuación se presentan las directivas más importantes de Vue.js [44].

`v-html`: Permite que un texto con etiquetas HTML sea renderizado, obteniendo como resultado, un texto final sin etiquetas HTML.

`v-show`: Permite ocultar una etiqueta HTML, y sus etiquetas anidadas, si el valor pasado a `v-show` es falso. De lo contrario, si el valor pasado a `v-show` es verdadero, entonces mostrará una etiqueta y sus etiquetas anidadas. Esta directiva únicamente lo que hace es agregar un estilo CSS del tipo `display:show` o `display:none`, según corresponda, para ocultar o mostrar elementos, por lo tanto, no son eliminados del DOM cuando `v-show` contiene el valor falso.

`v-if`: Es una directiva similar a la directiva `v-show`. Sin embargo, la diferencia radica en que no se aplica estilos, sino que la etiqueta HTML, y sus etiquetas anidadas, son eliminadas del DOM real. Sin embargo, son mantenidas en memoria en caso de requerir un estado anterior.

`v-model`: Crea un mecanismo de comunicación de dos vías para que la información que sea ingresada o manipulada en un *input* o incluso, un componente, también se vea reflejada en otro elemento. En el Código 1.24 se presenta un ejemplo de su uso.

```
<input v-model="message" placeholder="ingrese cualquier texto" />
<p>Mensaje: {{ message }}</p>
<script>
    new vue({
        data: {
            message: ''
        }
    })
</script>
```

Código 1.24. Ejemplo de uso de directiva `v-model` de Vue

El resultado del renderizado del Código 1.24, se puede observar en la Figura 1.12. Cualquier texto que se modifique dentro del *input*, se verá reflejado en la parte inferior. La directiva `v-model` puede ser usada para `inputs`, `checkboxes`, `textareas`, `radiobuttons`, etc.

Figura 1.12. Ejemplo del resultado HTML de usar directiva `v-model` de Vue

`v-for`: Es una directiva que provee la funcionalidad de recorrer estructuras como *arrays* u objetos Javascript con el propósito de renderizar todo el contenido que está dentro de dichas estructuras, en etiquetas HTML y obtener un resultado final.

En el Código 1.25, se muestra cómo utilizar la directiva `v-for`. Como se puede visualizar, dicha directiva se pasa como un atributo de la etiqueta div.

```
<div v-for="item in items">
    {{ item.text }}
</div>

<script>
    var items = ['uno', 'dos', 'tres']
    new vue({
        data: {
            items: items
        }
    })
</script>
```

Código 1.25. Ejemplo de uso de directiva `v-for` de Vue.js

El resultado final después de renderizar el código anterior se presenta en el Código 1.26.

```
1   <div>
2       uno
3   </div>
4   <div>
5       dos
6   </div>
```

Código 1.26. Resultado HTML después de renderizar usando `v-for`

`v-on`: Es una directiva usada para los eventos Javascript como `click`, `keyup`, `keydown`, etc. Se puede agregar de forma individual una acción o incluso, múltiples acciones, como se aprecia en el Código 1.27.

```
1   <div v-on:click="methodOnClick" ></div >
2   <div v-on="
3           click: methodOnClick,
4           keyup: methodOnKeyup,
5           keydown: methodOnKeydown">
6   </div >
```

Código 1.27. Formas de declarar directivas para Vue

i. Codemirror

Es una librería de Javascript que permite incrustar un editor de código en una página web. Provee una API para soportar sintaxis, resaltado e identación para una larga lista de lenguajes de programación utilizando CSS y Javascript.

El uso de esta librería resulta ser bastante sencillo pues para empezar a utilizarla, se debe especificar las rutas de las librerías de estilo CSS, las librerías de Javascript y pasar como argumentos al objeto Codemirror, el objeto DOM HTML donde será contenido dicho editor y, opcionalmente, algunos parámetros de configuración que permiten definir la sintaxis, el tema visual, la identación, el lenguaje de programación entre otros.

Como se muestra en el Código 1.28, en la línea 1 y 2 se especifica la ruta de las librerías CSS. La línea 1, corresponde al diseño base de Codemirror que provee una interfaz visual del editor de código y la segunda, corresponde al tema visual que provee un diseño más atractivo para el editor. La línea 4, corresponden a un contenedor `div` sobre el cual se insertará el editor de código. En la línea 6, se especifica la ruta del *script* que provee las funcionalidades básicas de un editor de código. Esto es, mostrar número de líneas, generar identación, cierre automático de llaves, entre otros. Para proveer funcionalidades como el resaltado de un lenguaje en particular, se especifica la ruta del lenguaje que se

desea asociar. La librería `clike` provee la sintaxis de resaltado para aquellos lenguajes que tienen una estructura similar. Por ejemplo, los lenguajes de programación Java y C#.

```
1   <link rel="stylesheet" href="lib/codemirror.css">
2   <link rel="stylesheet" href="theme/default.css">
3
4   <div id="container-codemirror"></div>
5
6   <script src="lib/codemirror.js"></script>
7   <script src="mode/clike/clike.js"></script>
8   <script>
9
10   var configuration = {};
11   var target = document.getElementById('container-codemirror');
12   Codemirror(target, configuration);
13
14  </script>
```

Código 1.28. Ejemplo de uso de Codemirror

j. Vee-Validate

Es un *plugin* para Vue.js el cual permitirá validar campos de un formulario HTML y mostrar errores. La ventaja de utilizar este *plugin* es que permite flexibilidad al agregar ciertas especificaciones como atributos a los campos de un formulario HTML. Además, permite agregar validaciones personalizadas utilizando expresiones regulares[31].

Vee-validate provee de forma predeterminada una serie de reglas para validación. Entre las más importantes se encuentran validación de números, texto, correo electrónico, número de caracteres mínimos y máximos, entre otros [45].

En el Código 1.29 se muestra un ejemplo de cómo importar este *plugin* para que pueda ser utilizado conjuntamente con Vue.js. La línea 3, utilizando el método `use` de `Vue`, permite que vee-validate pueda ser usado como un complemento de Vue.js.

```
1   import Vue from 'vue';
2   import VeeValidate from 'vee-validate';
3   Vue.use(VeeValidate);
```

Código 1.29. Ejemplo de importar plugin en instancia Vue

Para utilizar los patrones de validación de vee-validate se debe agregar como atributo en un campo de formulario HTML, la palabra clave `v-validate` y varios elementos de validación separados por una barra vertical, como se presenta en el Código 1.30.

[31] Expresión regular es una combinación de caracteres como patrones para realizar búsqueda parcial o total dentro de un *string* o cadena de caracteres.

```
<input type="text" v-validate="'validación-1|validación-2|…'">
```

Código 1.30. Declaración `v-validate` como atributo HTML

Como, por ejemplo, si se desea que un campo sea obligatorio y, además, de tipo correo electrónico, el *string* que se debe pasar al atributo `v-validate`, se muestra en el Código 1.31. Es importante mencionar que dicho *string* debe estar contenido entre comillas simples para que vee-validate pueda interpretar la información. Las comillas dobles corresponden al formato que se define para especificar valores dentro de un atributo HTML.

```
v-validate="'required|email'"
```

Código 1.31. Múltiples validaciones en el atributo `v-validate`

Si se requiere extender la funcionalidad de vee-validate para proveer nuevos patrones de validación, se debe agregar al *plugin*, cierta estructura de datos.

```
const nuevaValidacion = {
    getMessage(field, params, data) {
        return field + ' mensaje de error si el patrón no coincide'
    },
    validate(value) {
        // agregar la validación correspondiente ya sea por medio de
        // expresiones regulares o sentencias if/else
        return true/false
    }
}
VeeValidate.Validator.extend('nuevaValidacion', nuevaValidacion)
```

Código 1.32. Creación de nuevos patrones de validación en vee-validate

Como se presenta en el Código 1.32, al método `extend` se le debe pasar como primer argumento, el nombre del patrón de validación y, como segundo argumento, un objeto Javascript que contiene dos funciones:

- La primera función con nombre `getMessage`, la cual permitirá mostrar un mensaje de error cuando el patrón no coincide, acepta 3 argumentos:
    - el primero corresponde al identificador del campo HTML del formulario que se ha validado.
    - el segundo, corresponde a elementos de validación que fueron definidos en `v-validate`.

- el tercero, corresponde a los datos que fueron ingresados en el campo HTML.

- La segunda función con nombre `validate`, permitirá validar los valores que sean ingresados al campo del formulario HTML. Esta función recibe un argumento y corresponde al valor del campo HTML del formulario para que pueda ser evaluado y retorne `true` o `false` según corresponda.

k. Viper

Es una librería para Golang diseñada para proveer un mecanismo de configuración de variables de una aplicación a través de archivos, variables de entorno, línea de comandos, entre otros.

Provee la capacidad de trabajar con diferentes tipos de formatos como por ejemplo JSON. Para el presente proyecto, esta librería es utilizada para cargar algunas variables de configuración dentro del sistema tanto para el ambiente de desarrollo como para el ambiente de producción [46].

```
1   viper.Set("nombre_variable", "tipo_variable")
2
3   viper.GetBool("nombre_variable")
4   viper.GetInt("nombre_variable")
5   viper.GetString("nombre_variable")
```

Código 1.33. Ejemplo de uso de Viper para establecer y recuperar variables

En el Código 1.33, se presenta un ejemplo de cómo pueden ser establecidas algunas variables de entorno y cómo estas pueden ser recuperadas utilizando Viper.

Para definir variables de entorno se utiliza el comando que se muestra en la línea 1; el método `set` recibe dos parámetros: el primero corresponde al nombre de la variable que se desea crear y, el segundo, corresponde al valor que será asociado a la variable en cuestión.

El valor puede ser de tipos que son soportados por Golang como por ejemplo enteros, *strings*, booleanos, entre otros. Para obtener variables establecidas con el comando de la línea uno, se deben utilizar métodos que particularmente indican el tipo de dato que va a ser recuperado. Por ejemplo, en la línea 3, utilizando el método `GetBool`, se indica que se va a recuperar un dato de tipo `bool`. De igual manera en la línea 4, se recupera un dato de tipo `int` utilizando el método `GetInt`. El método `GetString`, mostrado en la línea 5, recuperará un dato de tipo `string`.

Los métodos para recuperar el valor de una variable que previamente ha sido establecida, aceptan un único parámetro que corresponde al nombre de la variable en cuestión.

Dado que se requiere que las variables de entorno que son utilizadas por el sistema, se configuren de forma rápida sin necesidad de realizar cambios en el código fuente de la aplicación, Viper provee una alternativa para establecer dichas variables a través de archivos de configuración. Para el presente proyecto, se utilizan archivos con extensión JSON para definir dichas variables.

En el Código 1.34, se muestra un ejemplo de un archivo JSON con nombre `config` que contiene una estructura de datos jerárquica.

```
{
    "host": {
        "address": "localhost",
        "port": 8000
    },
    "database": {
        "production": {
            "host": "127.0.0.1",
            "port": 8080
        },
        "development": {...}
    }
}
```

Código 1.34. Variables de entorno de ejemplo en una estructura JSON

En el Código 1.35, se muestra el proceso de leer el contenido del Código 1.34 y asignar, cada uno de los elementos de la estructura que contiene información, como variables de entorno de Go. En la primera línea, al método `SetConfigName`, se le pasa como parámetro el nombre del archivo de configuración con formato JSON, que se leerá para obtener la estructura de datos contenida en dicho archivo. En la segunda línea, al método `AddConfigPath`, se le pasa como parámetro la ruta donde se encuentra dicho archivo. Finalmente, en la tercera línea se llama al método `ReadInConfig` para que realice la tarea de leer el contenido del archivo cargado y asigne como variables de entorno, aquella información, de la estructura de datos, que será utilizada.

```
viper.SetConfigName("config")
viper.AddConfigPath("ruta_del_archivo_json")
viper.ReadInConfig()
```

Código 1.35. Establecer variables de entorno en un archivo JSON

Realizado el proceso mostrado en el Código 1.35, se podrá acceder al valor de cualquier dato especificado en el ejemplo del Código 1.34, teniendo en cuenta que, para obtener valores anidados dentro de la estructura JSON, se debe utilizar la notación punto por cada nivel que conforma dicha estructura. Por ejemplo, para acceder al valor `host` dentro de `production` y, este a su vez, dentro de `database`, de la estructura JSON de ejemplo, se debe usar el *string* `database.production.host` como se muestra en el Código 1.36.

```
1 viper.GetString("database.production.host")
```

Código 1.36. Ejemplo para obtener el valor de una estructura JSON jerárquica

I. Bone

Es una librería utilizada como un enrutador tanto para la aplicación web como para los servicios web. Tiene soporte para manejar parámetros a través de la URL, declaración de métodos HTTP, manejador de error 404 de forma personalizada, entre otras características [47].

Observando nuevamente al método de la línea 9, del Código 1.10, se verifica que se le pasa como argumentos, la ruta de acceso y el controlador asociado, sin embargo, no se define en ningún instante, que tipo de petición manejará (GET, POST, etc.).

La librería Bone permite definir de forma explícita el tipo de petición y el método del controlador que será utilizado para manejar dicha petición, como se aprecia en el Código 1.37. Además, para peticiones de tipo GET, Bone permite que se definan variables dinámicas que serán pasadas en el segmento de la URL, utilizando la notación `:nombre_variable`.

En la línea de 1 del Código 1.37, se obtiene una nueva instancia de `Bone` utilizando el método `New`. En la línea 2, se utiliza el método `Get` de la instancia creada y la cual recibe dos parámetros: el primero, corresponde a la URL `/inicio` con un segmento que contiene una variable dinámica llamada `id`; y el segundo, corresponde al nombre del método que manejará la petición GET asociada a la URL definida.

En la línea 3, se tiene un caso similar al ejemplo de la línea 2, con la diferencia de que, en la URL especificada, se tienen dos variables dinámicas llamadas `id` y `var`. En cambio, en el código de la línea 4, se utiliza el método POST, de la instancia creada, para manejar peticiones de tipo `Post`. Así mismo recibe dos parámetros: el primero, corresponde a la URL asociada al tipo de petición mencionada anteriormente y el

segundo, corresponde al método que manejará dicha petición. En este tipo de peticiones, no se especifica variables en la URL puesto que las variables son pasadas en el cuerpo del mensaje HTTP.

```
1 mux := bone.New()
2 mux.Get("/inicio/:id", http.HandlerFunc(InicioControlador.NombreMetodo))
3 mux.Get("/info/:id/:var", http.HandlerFunc(InfoContralador.NombreMetodo))
4 mux.Post("/panel", http.HandlerFunc(PanelControlador.NombreMetodo))
```

Código 1.37. Ejemplo de uso de la librería Bone para enrutamiento

Tomando como ejemplo el controlador asociado a la URL de la línea 2 del Código 1.37, para recuperar el valor dinámico `id` que está asociado a la URL `/inicio`, se utilizará el método `GetValue` que está contenida en la estructura de datos `http.Request` como se ve en la línea 7 del Código 1.38. Dicho método recibe dos parámetros: el primero corresponde a la estructura de datos de la petición asociada, que es pasada como segundo parámetro de la función `NombreMetodo`; y el segundo argumento, corresponde al nombre de la variable dinámica que está asociada a la URL en cuestión, esto es `/inicio/:id`. Dado que el valor que se pasa a la variable en cuestión es un *string*, dicho *string* puede tener cualquier contenido, es decir, enteros, estructuras de datos como JSON, etc.

```
1 package InicioControlador
2 import (
3     "net/http"
4     "github.com/go-zoo/bone"
5 )
6 func NombreMetodo(rw http.ResponseWriter, req *http.Request){
7     id := bone.GetValue(req, "id")
8 }
```

Código 1.38. Recuperar variable dinámica de una URL en una petición GET

1.3.14. JUnit y NUnit

Las librerías de pruebas unitarias contienen métodos de aserción. Estos métodos, suelen recibir dos parámetros: el primero corresponde a un valor esperado y el segundo, corresponde al valor del resultado obtenido después de haber ejecutado el método de una clase que se desea probar. Cuando el método de aserción compara el resultado esperado con el resultado obtenido, este devolverá `true` si la comparación resulta exitosa, caso contrario, retornará `false`. En cualquiera de los dos casos, la librería se encargará de generar un resumen de dicho resultado para ser mostrado en consola.

Puesto que el código para Java y C# pueden hasta cierto punto mantener una estructura común sin hacer uso de librerías externas o definición de paquetes para Java o espacios de nombre para C#, el Código 1.39 se utiliza para mostrar los ejemplos de uso de las librerías JUnit y CUnit.

```
1 Class Numeros {
2     public static int Sumar(int a, int b)
3     {
4         return a + b;
5     }
6 }
```

Código 1.39. Código de ejemplo para Java y C# para sumar números

El Código 1.40 muestra un ejemplo de uso de la librería JUnit para evaluar el Código 1.39. En las líneas de código 1 a 3, se importan las librerías de pruebas y la clase que será probada. Los métodos `pruebaUno`, `pruebaDos` y `pruebaTres`, de la clase `JavaTest` contienen una anotación que indica que dichos métodos serán ejecutados para llevar a cabo las pruebas. Como se puede visualizar en el Código 1.40, cada método de la clase `JavaTest` realizará las pruebas sobre el método `Sumar` de la clase `Numero`. Para ello se le pasa dos argumentos y el resultado obtenido se compara con el resultado esperado, a través del método `assertEquals`.

```
1  import static org.junit.Assert.assertEquals;
2  import org.junit.Test;
3  import Numeros;
4  public class JavaTest {
5      @Test
6      void pruebaUno() {
7          int prueba1 = Numeros.Sumar(1, 1);
8          assertEquals(2, prueba1);
9      }
10     @Test
11     void pruebaDos() {
12         int prueba2 = Numeros.Sumar(5, 10);
13         assertEquals(-35, prueba2);
14     }
15     @Test
16     void pruebaTres() {
17         int prueba3 = Numeros.Sumar(-15, -10);
18         assertEquals(15, prueba3);
19     }
20 }
```

Código 1.40. Ejemplo de uso de JUnit

El mismo proceso se realiza para la evaluación con NUnit, la diferencia radica en la importación de librerías, en la forma en cómo se declara la anotación para indicar que el método es de prueba y en los métodos de aserción para evaluar el método de una clase.

En el Código 1.41, se muestra un ejemplo del uso de NUnit, así mismo, utilizando el Código 1.39 de ejemplo. En la línea 1, se importa la librería NUnit, en la línea 2, se importa la clase que va a ser probada. En la línea 3, se declara una anotación que indica que la clase que se crea corresponde a una clase de pruebas unitarias. Como se pueden ver las anotaciones declaradas en las líneas 6, 12 y 18, se encuentran encerradas entre corchetes y en cada método se especifica tanto el método de la clase que se desea evaluar, pasándole argumentos de prueba, y comparando el resultado obtenido con el resultado esperado usando el método estático `AreEqual` de la clase `Assert`.

```
1  using NUnit.Framework;
2  using Numeros;
3  [TestFixture]
4  public class CSharpTest
5  {
6      [Test]
7      void pruebaUno()
8      {
9          int prueba1 = Numeros.Sumar(1, 1);
10         Assert.AreEqual(2, prueba1);
11     }
12     [Test]
13     void pruebaDos()
14     {
15         int prueba2 = Numeros.Sumar(5, 10);
16         Assert.AreEqual(-35, prueba2);
17     }
18     [Test]
19     void pruebaTres()
20     {
21         int prueba3 = Numeros.Sumar(-15, -10);
22         Assert.AreEqual(15, prueba3);
23     }
24 }
```

Código 1.41. Ejemplo de uso de NUnit

1.3.15. Ambientes de trabajo

En el desarrollo de software, es de vital importancia disponer de varios entornos de trabajo pues estos permitirán no solo realizar configuraciones propias para cada entorno, sino también, realizar actividades como pruebas de funcionalidad y rendimiento, entre otras.

Además, no se puede considerar que los recursos que son utilizados en el ambiente de desarrollo, necesariamente serán los mismos en el ambiente de producción. Podría ser que dichos recursos puedan ser sub-utilizados o sobre-utilizados [50].

A continuación, se detallarán dos ambientes de trabajo, pueden existir más, sin embargo, en lo referente al tema propuesto, estos son los más importantes que se han tomado en cuenta en todo el proceso de implementación del prototipo.

a. Desarrollo

Generalmente es un ambiente local configurado en un equipo personal. Toda la implementación se lleva a cabo en este ambiente y, por lo tanto, se tienen configuraciones locales propias para este ambiente. Las configuraciones están asociadas a variables que contienen información sensible de conexión a base de datos, a servicios de terceros para correo electrónico, puertos, entre otros.

Es importante mencionar que este tipo de ambientes no requiere de una capacidad excesivamente alta, pues el desarrollo y las pruebas que se van realizando en este ambiente no demandan de muchos recursos tanto de memoria como de consumo de procesamiento.

Sin embargo, por lo general se cuenta con un equipo con altas prestaciones en procesamiento y memoria, por lo tanto, este detalle se debe tener en cuenta puesto que en este ambiente el trabajo de la aplicación presenta un flujo normal y eficiente, sin embargo, en un ambiente de producción por lo general los recursos son de menor capacidad pues dependerá de cuantos usuarios y la concurrencia que manejará.

b. Producción

Este ambiente de trabajo está más enfocado en disponer de un entorno adecuado para poner el sistema en funcionamiento, teniendo en consideración que no solo será un usuario el que acceda a dicha aplicación sino una mayor cantidad. Por lo tanto, es importante considerar algunos detalles en el instante de codificar la aplicación. Por ejemplo, las configuraciones asociadas a este ambiente son totalmente diferentes a las configuraciones realizadas en un ambiente de desarrollo.

Por lo general, en ambientes de desarrollo la aplicación carga ciertos parámetros de configuración a través de archivos que contienen dicha configuración. Sin embargo, en ambientes de producción la situación es diferente pues, generalmente, dichos parámetros de configuración son pasados como variables de entorno. Esto depende en cierta medida de si se va a utilizar servicios en la nube o una infraestructura propia. Con frecuencia, los

servicios en la nube proveen un mecanismo de utilizar variables de entorno para establecer los parámetros de configuración.

1.3.16. Herramientas de desarrollo

a. Visual Studio Code

Visual Studio Code también conocida como VCode, es un editor de texto desarrollado por Microsoft y ofrece toda la capacidad y funcionalidad de un entorno de desarrollo integrado (por sus siglas en inglés IDE), a través de la instalación de *plugins*. Es multiplataforma e incluye funcionalidades para soportar varios lenguajes de programación, integración con GIT, completar código de forma inteligente (basado en sugerencias), provee un terminal integrado, multi-pestañas de forma vertical y horizontal entre muchas otras buenas características [53].

Aunque es un editor que consume unos cuantos megas de memoria RAM, pero mucho menos que un IDE, su versatilidad y usabilidad es bastante buena para ser utilizado como una herramienta en todo el flujo de desarrollo de software.

b. Node Package Manager (npm)

Es un gestor de paquetes Javascript. Permite que todas las dependencias de un proyecto sean descargados desde un repositorio con el mismo nombre. Este gestor de paquetes viene con herramientas que permiten no solo descargar dichas dependencias sino también ejecutar *scripts*. Toda la descripción que necesita manejar npm es a través de un archivo con formato JSON donde se especifican valores de paquetes tanto de desarrollo como de producción, así como información acerca del autor y una descripción general del proyecto [55].

Por defecto, npm buscará el archivo `package.json` para agregar información que está relacionada con detalles acerca de datos importantes de la aplicación o módulo que se esté construyendo, así como información de dependencias para cada uno de los ambientes.

Algunos de los comandos más importantes se presentan en el Código Código 1.42.

```
1   npm init
2   npm <install/uninstall> -g <paquete1 paquete2 … paquete-n>
3   npm <install/uninstall> -save <paquete1 paquete2 … paquete-n>
4   npm <install/uninstall> -save-dev <paquete1 paquete2 … paquete-n>
5   npm start
```

Código 1.42. Lista de comandos más importantes de npm

Al ejecutar el comando de la línea 1, npm solicitará cierta información para posteriormente crear el archivo requerido para el proyecto `package.json`. Los comandos de la línea 2, la línea 3 y la línea 4 permitirán instalar o desinstalar las dependencias del proyecto.

La diferencia radica en cómo estos serán registrados en el archivo `package.json`. Si se utiliza la opción `-g`, la dependencia será instalada o desinstalada de forma global en el registro de paquetes de npm (carpeta del sistema `node_modules`) pero no mantendrá ningún registro de este suceso en el archivo `package.json`, que forma parte del proyecto. La opción `-save` permitirá que el paquete sea registrado como una dependencia para todos los ambientes de trabajo. La opción `-save-dev` permitirá registrar la dependencia que solo será utilizada para el ambiente de desarrollo. El comando de la línea 5, por lo general es configurado para inicializar un servidor Node.js[32] previamente configurado para empezar con el desarrollo de un módulo o aplicación.

c. DEP

DEP es un gestor de paquetes para el ecosistema de Golang. Al igual que npm, DEP permite realizar varias configuraciones sobre un archivo con extensión YAML[33] el cuál será utilizado por DEP para realizar tareas no solo con instalación de dependencias sino también con ejecución de *scripts*, mostrar información, entre otros. DEP buscará en la ruta principal del proyecto un archivo con nombre `Gopkg` y extensión `yaml` pues en dicho archivo se especifican todas las dependencias que el proyecto en cuestión necesitará para poder ser ejecutada.

En el Código 1.43, se muestran 3 comandos que con frecuencia son utilizados para gestionar un proyecto en Golang. El comando de la línea 1, permite instalar una dependencia en el proyecto. El comando de la línea 2, permite que se instalen todas las dependencias del proyecto, leyendo el archivo `Gopkg.yml`. El comando de la línea 3 permite actualizar todas las dependencias del proyecto.

```
1 dep ensure -add nombre_del_paquete
2 dep ensure
3 dep update
```

Código 1.43. Comandos para el gestor de paquetes DEP para Golang

[32] Node.js es un entorno de ejecución de Javascript. Este entorno permite que Javascript pueda ser ejecutado como un lenguaje en el lado del *backend*.
[33] YAML (*Yet Another Markup Language*) es un lenguaje de serialización de datos y utiliza identación en lugar de caracteres especiales para generar bloques informativos basados en una taxonomía jerárquica.

d. Vue-webpack-boilerplate

Es una herramienta para prototipado rápido, que permite generar una estructura base de Vue y todas las dependencias necesarias para empezar a desarrollar ya sea un componente web o la vista de MVC, de una aplicación web. Provee algunas características que permiten que el trabajo sea eficiente. Por ejemplo, cuenta con configuraciones para poder desarrollar sobre diferentes ambientes de trabajo: desarrollo, producción, entre otros.

Para el modo de desarrollo, cuenta con un servidor Node.js para realizar pruebas entre el componente y dicho servidor, además cuenta con un mecanismo llamado *hot-reload* el cual permite un continuo desarrollo sin la necesidad de refrescar el navegador por cada cambio que se realice en el código.

Por otro lado, el modo de producción permitirá generar recursos estáticos minificados y comprimidos para poder ser distribuidos como componentes independientes y se puedan agregar sobre cualquier aplicación web.

Para poder utilizar esta herramienta se ejecutarán en la línea de comandos las sentencias presentadas en el Código 1.44.

```
1   npm install -g vue-cli
2   vue init webpack my-project
3   cd my-project
4   npm install
5   npm run dev
```

Código 1.44. Comandos para instalar Vue-webpack-boilerplate

La línea 1, instalará una utilidad que permite que, a través de comandos sobre el terminal, se puedan ejecutar acciones para Vue. La línea 2 generará una estructura utilizando la utilidad Vue-webpack-boilerplate. Después de creada la estructura, se debe ingresar a la carpeta donde se ha creado el proyecto (línea 3) y, a continuación, instalar todas las dependencias de la estructura generada utilizando el comando de la línea 4. Finalmente, el comando de la línea 5 ejecutará la utilidad para empezar a desarrollar, luego de lo cual se mostrará de forma automática el proyecto en el navegador web.

La estructura generada se puede visualizar en la Figura 1.13. Puesto que lo más importante es empezar a desarrollar componentes, además de que todas las configuraciones para cada ambiente ya vienen pre-configuradas, las únicas carpetas que interesan son `static` y `src` pues, sobre la carpeta `static` se obtendrán los archivos

finales para producción mientras que la carpeta `src` contendrá aquellos archivos que permitirán desarrollar los componentes.

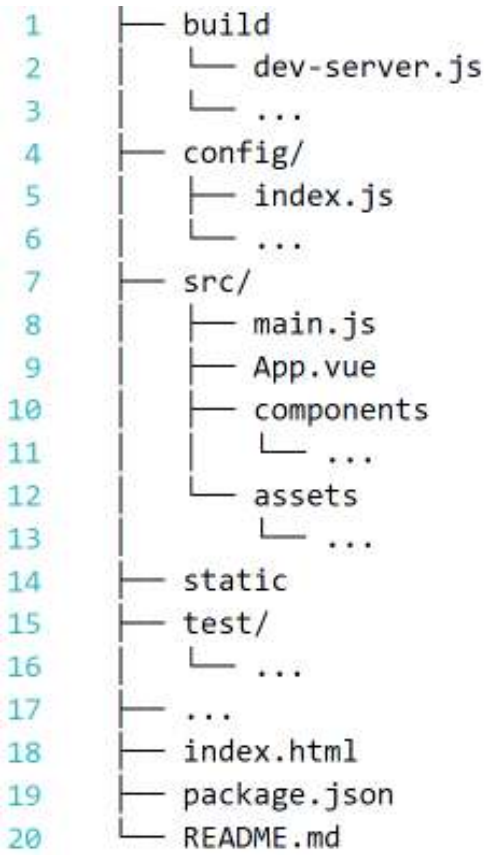


Figura 1.13. Estructura generada por Vue-webpack-boilerplate

En el Código 1.45, se presentan los comandos más importantes de esta utilidad. Mientras que el comando de la línea 1 permite ejecutar un servidor en *background* para empezar a desarrollar nuevos componentes, el comando de la línea 2 generará los archivos finales para ser integrados en cualquier aplicación web.

```
1  npm run dev
2  npm run build
```

Código 1.45. Comandos importantes de Vue-webpack-boilerplate

e. GitLab

Es una plataforma de servicio en la nube para proveer un sistema de control de versiones[34] basados en GIT[35]. Esta plataforma es de código abierto, y provee un

[34] Un sistema de control de versiones es un sistema que permite dar seguimiento y mantener registros, a lo largo del tiempo, acerca de los cambios que se realizan sobre un archivo o conjunto de archivos (repositorio)

sinnúmero de beneficios como administración de repositorios vía línea de comandos utilizando SSH[36], contar con un sistema de seguimiento de incidencias, crear y mantener repositorios totalmente privados, entre otras [51].

Esta plataforma será utilizada para gestionar el despliegue de la aplicación y los servicios web e incluso, descargar archivos de configuración y acceso al repositorio GIT de cada aplicación.

f. Amazon Web Services (AWS)

Es una plataforma de servicios en la nube provista por Amazon. Todos los servicios que ofrece AWS tienen un modelo de *Pay-As-You-Go* el cual indica que se paga solo por los recursos utilizados.

Algunos de los servicios que ofrece AWS y que serán utilizados en la puesta a producción del prototipo son [52]:

- *Amazon Elastic Compute Cloud* (EC2): Permite crear servidores usando máquinas virtuales. AWS lo llama una instancia y se puede realizar cualquier cosa sobre dicha instancia, así como lanzar n instancias.
- *Simple Storage Service* (S3): Capacidad de almacenamiento de archivos y registros.
- *Relational Database Service* (RDS): Administración de datos y base de datos. Particularmente se utilizará para gestionar una base de datos MySQL.
- *Elastic BeanStalk*: Es una plataforma como servicio y una capa de abstracción sobre EC2 que permite a los desarrolladores desplegar y administrar fácilmente cualquier aplicación web, pues ElasticBeanStalk automáticamente manejará todos los detalles de provisionamiento de componentes y servicios, así como la configuración y capacidad de computo de la infraestructura de EC2. ElasticBeanStalk se encargará, por ejemplo, de la instalación y configuración del sistema operativo creación de grupos de seguridad, manejo de variables de entorno, configuración de balanceadores de carga, configuración de archivos para el servidor, instalación de dependencias para los lenguajes de programación soportados, entre otros.

[35] GIT es un software creado para proveer un sistema de control de versiones.
[36] SSH por sus siglas en inglés (*Security Shell*) es un protocolo de comunicación que permite acceder desde un terminal Linux local, a través de una red insegura a un terminal Linux remoto utilizando mecanismos varios mecanismos de seguridad para mantener una sesión lo suficientemente confiable y segura.

1.3.17. Metodología ágil Kanban

La metodología de desarrollo ágil Kanban es frecuentemente utilizada como una herramienta que permite mantener un enfoque de las actividades que se deben desarrollar en un proyecto. Lo hace a través del denominado sistema de tarjetas (Figura 1.14).

En el desarrollo de software, generalmente cada tarjeta representa un componente o módulo que debe ser desarrollado. A cada tarjeta se le agrega una descripción lo más simple posible pero que permita dar una idea general de que se desea construir. Esta información no es técnica pero si puede incluir por ejemplo: asignación de responsabilidades, nivel de complejidad u otros datos que el equipo de desarrollo pueda considerar importante adjuntar [58].

El tablero más básico de Kanban está compuesto por varias columnas. Por ejemplo, una columna que identifica las actividades que se van a realizar (*To do*), otra columna que contendrá las actividades que están en curso (*In progress*), y una columna sobre la cual se agregan las tarjetas cuyas actividades han sido concluidas (*Done*).

Además, su método de trabajo se basa en la iteración de cada una de las tarjetas a lo largo de las columnas que conforman el tablero Kanban. Algo muy importante que se debe mencionar es que no existen restricciones o reglas que determinen cuantas y cuales columnas se deben utilizar ni tampoco, cuáles y cuantos son los elementos que se deben agregar a la tarjeta para proveer una información detallada. No obstante, el flujo de trabajo, la información y el número de columnas dependerá del tamaño del equipo de trabajo, de la complejidad del sistema, entre otros factores.

Además, se pueden utilizar tantos tableros como se requiera para diferentes etapas en el desarrollo de un producto. Esto con el propósito de distribuir de mejor manera las actividades en cada etapa [58].

Dado que, para la presente implementación, no existe un equipo de trabajo sino un único desarrollador, se considerarán 3 tableros Kanban para las etapas de Análisis, Diseño e Implementación del prototipo. Cada una de estas tarjetas permitirá mantener el enfoque de las actividades que se deben desarrollar en cada etapa. Además, permitirá dar una merjor perspectiva del trabajo que se debejo completar.

Como se puede visualizar en la Figura 1.14, hay un flujo continuo a través de cada una de las columnas de cada una de las tarjetas. Este proceso es iterativo hasta terminar la totalidad de actividades que deben ser completadas.

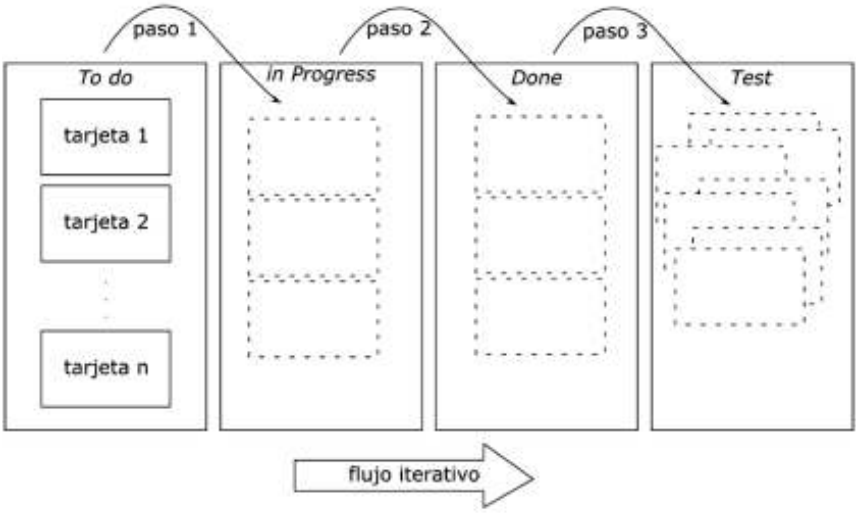


Figura 1.14. Tablero Kanban

2 METODOLOGÍA

El presente Trabajo de Titulación tiene como tal un enfoque aplicado dado que se realiza la implementación de una aplicación de software que está particularmente orientada a proveer un sistema para torneos de programación.

En esta sección, hay diferentes etapas que ayudarán a obtener el prototipo propuesto. Estas etapas tienen que ver con el análisis, diseño, implementación y despliegue del prototipo.

Sin embargo, previo a entrar en detalles en cada una de dichas etapas, se establecen un conjunto de tareas, utilizando el tablero Kanban, que se deben llevar a cabo para cumplir con el objetivo de cada etapa.

En la primera etapa, la etapa de análisis, se obtienen los requerimientos del sistema. Para ello, se recolecta información a través de la observación del funcionamiento de alguna aplicación que tiene el mismo enfoque que el tema propuesto.

Esta aplicación es la plataforma web CodeFights y permitirá que se obtengan ciertos requisitos tanto funcionales como no funcionales y, además, una mejor comprensión de cómo será la interfaz gráfica y los elementos que lo componen, para disponer del sistema de torneos de programación.

Con esta información adquirida, en la segunda etapa se procede a modelar, a través de diagramas UML, el comportamiento que tendrán cada uno de los requerimientos obtenidos. Esto con el propósito de tener un panorama mucho más claro de lo que realmente se va a implementar y como estos requerimientos deberán presentar un flujo acorde a lo que se requiere para el prototipo.

Posterior a la segunda etapa, en la etapa de implementación, se desarrollarán cada uno de los componentes de tal manera que cada uno de los requisitos sean cumplidos acorde a la información obtenida y los diagramas realizados.

Finalmente, esta implementación realizada en un ambiente de desarrollo, será pasada a un ambiente de producción de tal manera que permita realizar las pruebas correspondientes para determinar la funcionalidad del sistema y concluir que dicho sistema cumple con los requerimientos y objetivos planteados para este proyecto de titulación.

2.1 Análisis

Antes de empezar con el proceso de análisis, se definen, en el tablero Kanban, algunas actividades que se van a llevar a cabo. Dichas actividades se muestran en la Figura 2.1. Las cuatro primeras tarjetas del tablero Kanban, están asociadas a realizar un análisis sobre la plataforma web CodeFights. La última tarjeta está enfocada en la actividad de obtener requerimientos adicionales necesarios para poder implementar el prototipo propuesto.

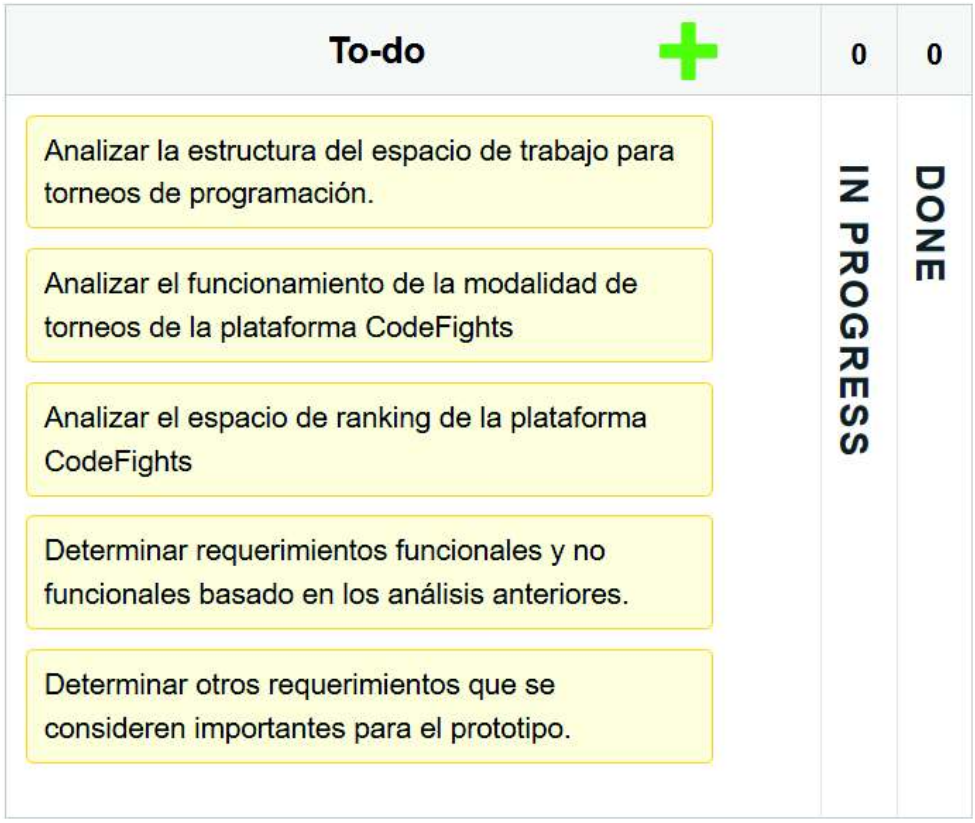


Figura 2.1. Tablero Kanban - Definición de tareas para la etapa de análisis

También es importante dar un contexto más amplio de la plataforma CodeFights que va a ser analizada.

CodeFights fue fundada por Trigan Sloya y es una aplicación web que provee un sistema para resolver problemas de programación utilizando diferentes modalidades de juegos en línea. La idea fundamental de esta plataforma está enmarcada en ser utilizada como una plataforma de competencias la cual permite que el aprendizaje de programación sea mucho más fácil y divertido [59].

Codefights presenta varias modalidades de juego, sin embargo, la modalidad de juego en línea para torneos de programación es la que será analizada.

2.1.1. Análisis del área de torneos de programación de CodeFights

La Figura 2.2, muestra cómo es el área que se utiliza para resolver problemas propuestos de programación. En el área, se puede visualizar las secciones más representativas del espacio de torneos. A continuación, se describen dichas secciones.

En la parte derecha superior, se puede visualizar un espacio similar a un editor de texto con detalles de resaltado e identación para cada lenguaje de programación.

En la parte derecha inferior existen dos espacios: el primero provee un conjunto de pruebas, mientras que el segundo espacio, provee un área para mostrar el resultado final cuando se haya evaluado el código enviado por el competidor.

En la parte izquierda, un bloque que provee información acerca del problema propuesto. Se puede visualizar que el contenido es del tipo de formato Markdown [37].

En la parte izquierda inferior, existen un conjunto de botones que permiten evaluar y enviar el código.

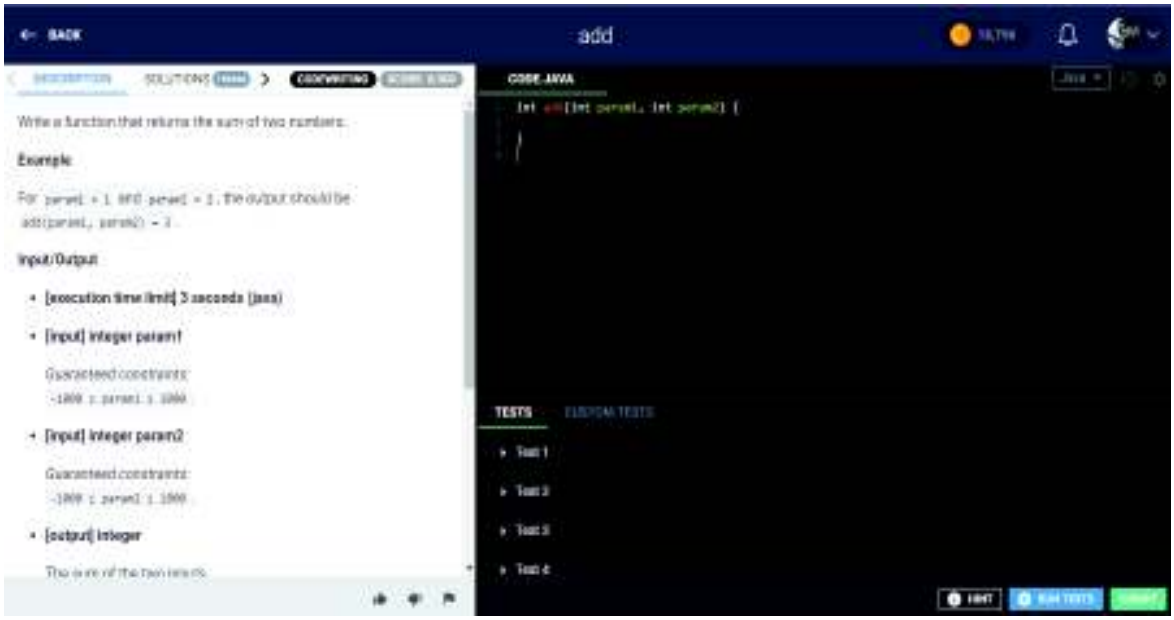


Figura 2.2. Interfaz gráfica para torneos de programación de una web

Es importante mencionar que cualquier usuario registrado en el sistema puede crear un torneo de programación. A continuación, se enumera el proceso de cómo trabaja CodeFights para crear torneos de programación:

1. Se accede a la página web con una cuenta de usuario previamente creada.

[37] Markdown ha sido creado como un lenguaje de marcado para proveer una forma fácil y legible de escribir texto sin hacer uso de etiquetas como en otros lenguajes de marcado como HTML.

2. Se selecciona, en la parte superior derecha la sección *Tournaments*. Será redireccionado a una nueva página donde podrá ver una lista de torneos que se han llevado a cabo como se muestra en la Figura 2.3.

3. Se presiona el botón *Create* (Figura 2.3) para poder crear un nuevo torneo. Posteriormente, cuando se haya presionado dicho botón, se desplegará un cuadro de diálogo con un formulario cómo se ve en la Figura 2.4. En la parte inferior del cuadro de diálogo se puede identificar que existen varias etapas que deben ser completadas para crear el evento en cuestión.

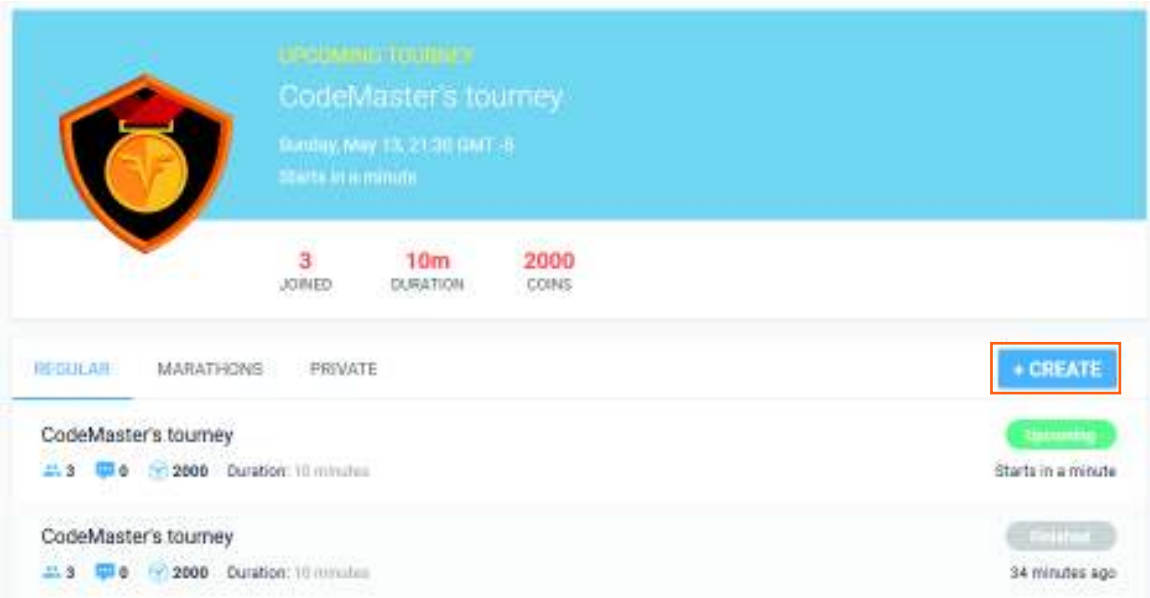


Figura 2.3. Página de torneos CodeFights

4. Después de presionar el botón *Next* (Figura 2.4), podrá visualizar un espacio para agregar los problemas propuestos que van a formar parte del torneo.

Figura 2.4. Etapa 1 - CodeFights – Crear un nuevo torneo

5. Como se observa en la Figura 2.5, se pueden elegir dos opciones: crear problemas personalizados o alteatorios. Se ha seleccionado la opción *Custom*. El cual mostrará un espacio de trabajo como se muestra en la Figura 2.6, sobre el cuál se pueden agregar nuevos problemas.

Figura 2.5. Etapa 2 – CodeFights – Agregando problemas propuestos

Figura 2.6. Etapa 3 – CodeFights – Agregar problemas personalizados

6. Cuando se presiona *Create new*, CodeFights abrirá una nueva página para agregar un nuevo problema. Como se puede ver en la Figura 2.7, el espacio para crear problemas cuenta con varias secciones entre las cuales se puede identificar:

   - En la parte inferior izquierda, una sección para agregar el valor esperado: incluye el tipo de variable y la descripción.
   - En la parte superior derecha, un espacio no editable sobre el cual se va generando código resaltado de forma automática basado en la información que se va agregando en el espacio correspondiente.

- En la parte inferior derecha, una sección para crear casos de prueba. Cuando se presiona en el botón *Add Test*, CodeFights desplegará, en la misma página, una ventana con un formulario que debe ser llenado con información, la cual describirá el caso de prueba (Figura 2.8).

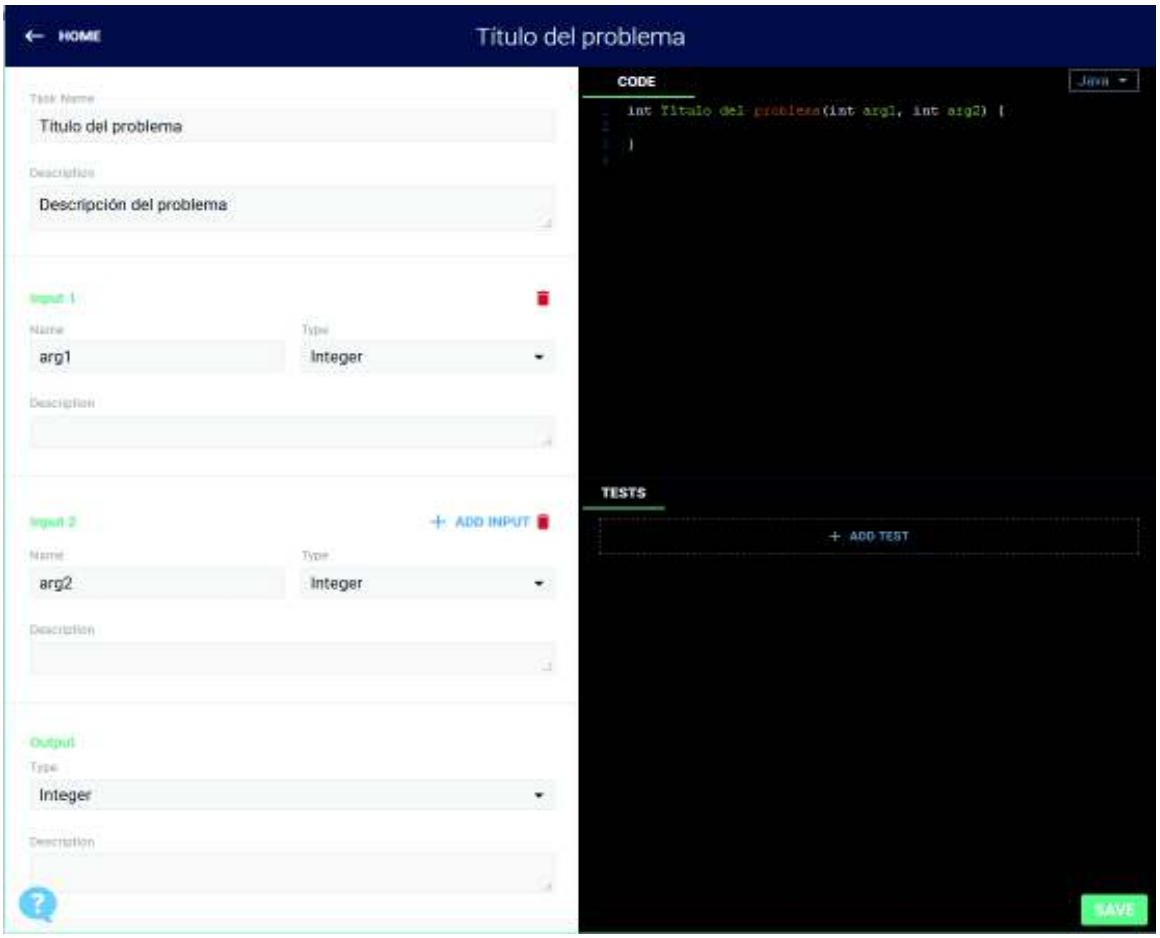


Figura 2.7. Etapa 3.1 – CodeFights – Agregar problema personalizado

Figura 2.8. Etapa 3.2 – CodeFights - Agregando casos de prueba

- Posteriormente, después de haber creado el problema, este se mostrará y podrá ser seleccionado (Figura 2.9). Al presionar el botón *Next* se pasará a la etapa final.

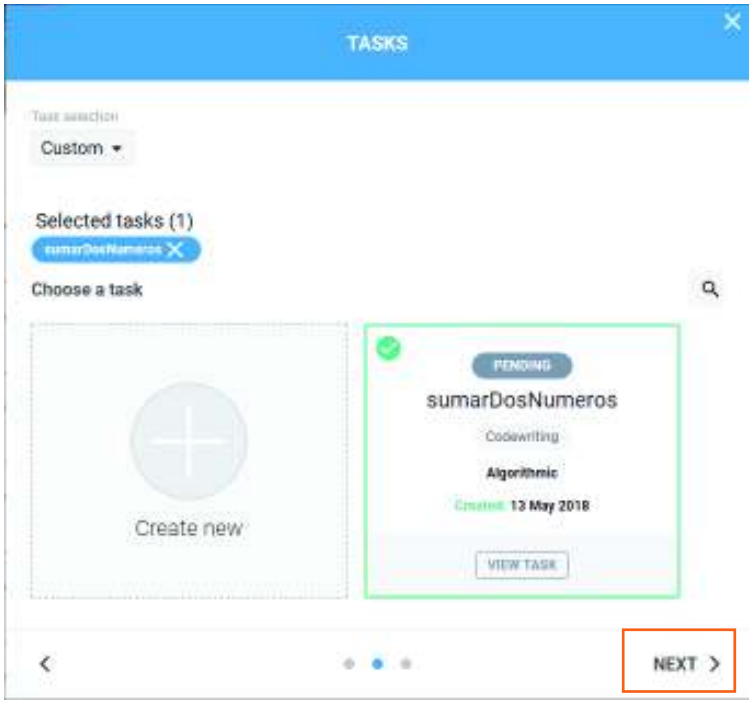


Figura 2.9. Etapa 3.3 – CodeFights - Agregar problemas personalizados

- En la etapa final, se podrá seleccionar la fecha y la hora en la que se llevará a cabo el torneo creado, así como también, el tiempo que durará dicho torneo (Figura 2.10). Finalmente, al presionar el botón *Done* se habrá creado el torneo y será público para aquellos usuarios que desean participar en dicho torneo.

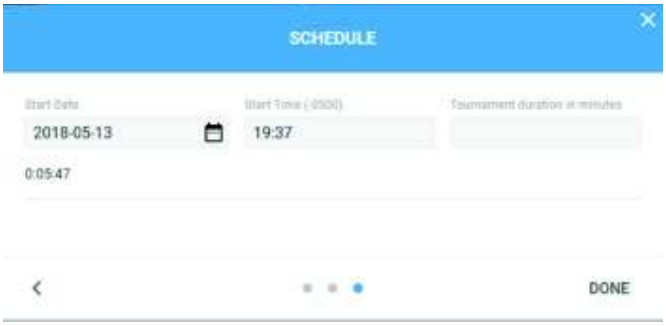


Figura 2.10. Etapa 4 – CodeFights – Seleccionar fecha para el torneo

2.1.2. Análisis de la página de competidores en CodeFights

En la Figura 2.11, se muestra una lista de usuarios según los puntos acumulados. Esta lista corresponde a un *ranking* general del sistema de CodeFights el cual se muestran los

puntos que han adquirido los usuarios en cualquiera de las modalidades de juegos en línea disponibles. La distribución de posiciones está dada por un enfoque visual destacable para los 3 primeros competidores con mayor puntuación. El resto de competidores se encuentra registrado en una tabla según el orden de puntuación.

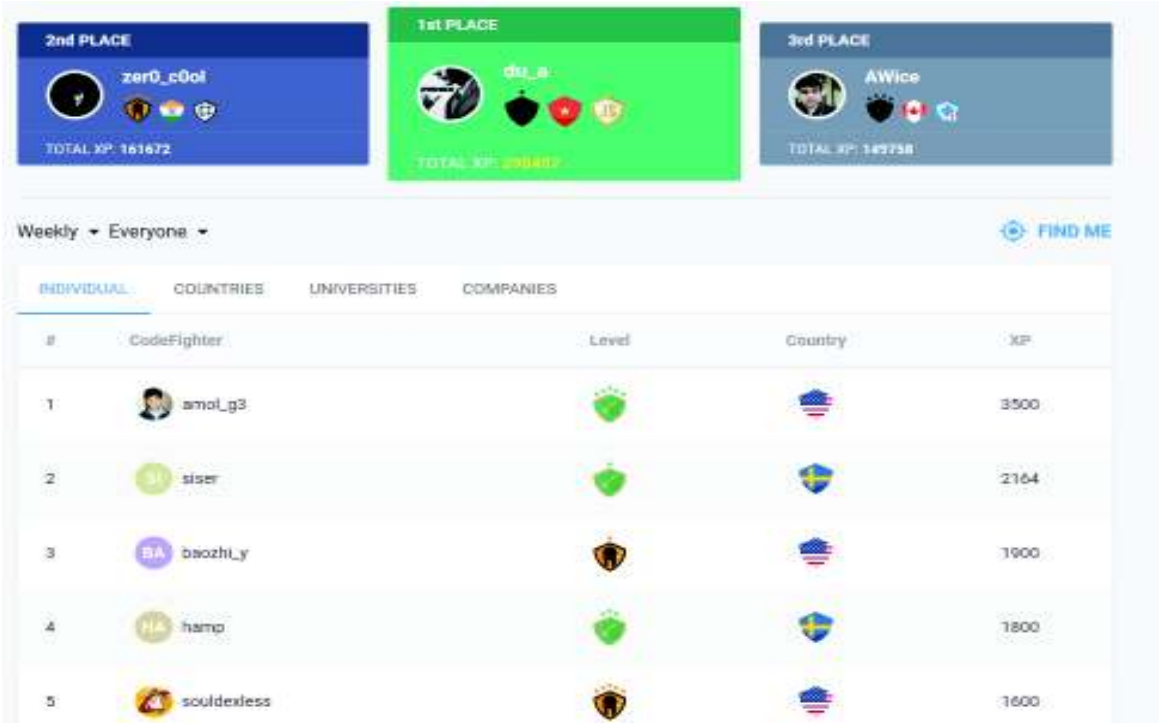


Figura 2.11. Tablero de posiciones de competidores en CodeFights

2.1.3. Requerimientos del sistema

Después de haber realizado un análisis de algunas secciones de CodeFights en la modalidad de torneos, se han podido identificar varias características para definir no solo requerimientos funcionales, sino también, requerimientos no funcionales y que serán implementadas en el prototipo propuesto.

a. <u>Requerimientos no funcionales</u>

- Disponer de un editor de código en línea a través del navegador web; editor similar a un IDE o editor de texto que soporte al menos dos lenguajes de programación.
- Los lenguajes de programación que soporta el prototipo serán Java y C#.
- Proporcionar una sección que permita mostrar la descripción del problema planteado.

- Proporcionar una sección que permita mostrar las reglas que están asociadas a un torneo.
- Disponer de un sistema de *ranking* de equipos competidores.
- Disponer de un tablero de administración para la gestión del sistema.
- Proporcionar un espacio que permita crear nuevos problemas con casos de prueba.
- Disponer de un área para actualizar datos de usuario.

b. Requerimientos funcionales

Los siguientes requerimientos están dados no solo por las características que han sido analizadas en la plataforma CodeFights sino también, por una retroalimentación dada durante el desarrollo del proyecto y otras que han surgido durante el análisis y desarrollo del presente proyecto. A continuación, se listan estos requerimientos.

- De los usuarios:
    - Permitir registrar nuevos usuarios en el sistema utilizando un correo electrónico. Los usuarios deberán validar su cuenta de correo electrónico a través del envío de un enlace de confirmación.
    - Permitir iniciar sesión solo a aquellos usuarios que han validado su cuenta.
    - Permitir cerrar sesión de aquellos usuarios que hayan iniciado sesión en su cuenta.
    - Permitir recuperar la contraseña si un usuario registrado en el sistema, olvida la misma. Para lo cual se utilizará el envío de un correo electrónico con un enlace que accede a un área para poder cambiar la contraseña.
    - Cada usuario registrado en el sistema, podrá llenar información acerca de su nombre y apellidos, universidad, carrera, nombre de usuario que será utilizado por el sistema y apodo que será utilizado para los eventos de competición.
- De los torneos y miembros de equipos participantes:
    - Cada torneo creado, deberá contener la fecha del evento, la hora, el número de participantes por equipo y las reglas del evento.

- Solo aquellos usuarios registrados y que han verificado su cuenta en el sistema, podrán crear un equipo para participar.
- Se podrán crear equipos únicamente cuando un torneo se haya organizado.
- Los usuarios podrán crear un único equipo para cada torneo.
- Un usuario autenticado, podrá invitar a usuarios a ser parte de un equipo previamente creado. Este proceso se realizará a través del envío de un correo electrónico de notificación al usuario invitado.
- Solo aquellos usuarios invitados que confirmen el correo electrónico y, posteriormente, llenen de forma obligatoria información de datos básicos, formarán parte del equipo.
- Los usuarios invitados no tienen acceso al sistema como un usuario registrado.
- Finalizado el tiempo de competición, el sistema deberá automáticamente enviar el código que el usuario haya escrito en el editor de texto y ser evaluado para emitir un resultado.

- Del área para resolver los problemas:
  - Se debe disponer de un temporizador en el área para resolver los problemas propuestos.
  - Permitir navegar a través de los problemas propuestos sin que el sistema se bloquee.
  - Después de concluido la resolución de problemas propuestos, se debe redireccionar a la página de *ranking* del torneo en cuestión.
- Del área de administración:
  - Permitir registrar de nuevas universidades y carreras.
  - Permitir registrar nuevos problemas propuestos.
  - Permitir registrar nuevos torneos teniendo en consideración lo siguiente:
    - Para crear un nuevo torneo, se debe elegir de una lista de problemas previamente creados. Si no existen problemas propuestos, no se permitirá el registro de un torneo.

        - Los problemas propuestos serán categorizados en tres niveles de dificultad y podrán ser reutilizados para cualquier torneo.
    - Ver listas de información acerca de problemas propuestos, de usuarios, de universidades, de carreras, de torneos y de equipos.

- De la evaluación de código:
    - Ejecutar tres o menos casos de prueba, sobre la solución de código de un problema planteado y que ha sido enviado por el equipo competidor.
    - El tiempo de prueba para cada código deberá ser ejecutado en un máximo de 3 segundos para evitar bloqueos, desbordamiento de memoria y uso excesivo de recursos.

- Del panel de usuario:
    - Debe tener una sección para cambiar su información
    - Se mostrará cualquier mensaje permitiendo o bloqueando el acceso al área de competición y creación de equipos teniendo en cuenta las consideraciones de los torneos y los equipos.

- Del área de *ranking*:
    - Cualquier usuario, registrado o no en el sistema, podrá ver el panel de *ranking*.
    - En la página principal se mostrará un enlace al *ranking* de la última competición.

2.2 Diseño

A continuación, se definen las tareas que serán desarrolladas en la etapa de diseño. El tablero Kanban quedaría establecido como se muestra en la Figura 2.12, y es donde se puede observar todas las tareas que se deben llevar a cabo para completar la etapa de diseño.

Como parte fundamental del desarrollo de software, se debe definir con un mayor grado de precisión, como serán las actividades y flujos de información que el sistema deberá llevar a cabo para cumplir con los requisitos funcionales y no funcionales especificados anteriormente. Para ello, es necesario realizar una serie de diagramas UML que permitan establecer dichos comportamientos.

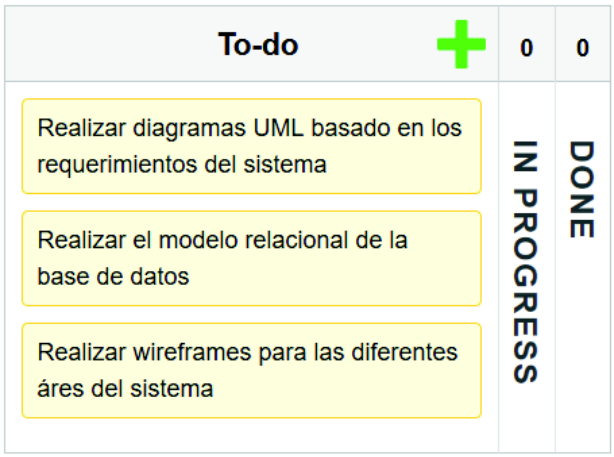


Figura 2.12. Tablero Kanban - Definición de tareas para la etapa de diseño

2.2.1. Diagrama de caso de uso

En primera instancia, se deben identificar los actores del sistema y como estos interactúan con el mismo. Uno de los diagramas que permite la interacción entre los actores y sistema, es el diagrama de casos de uso.

Basado en los requerimientos especificados anteriormente, se generó el diagrama UML que se muestra en la Figura 2.13. El diagrama UML de casos de uso muestra un contexto general de cómo cada uno de los actores externos tendrá una interacción con el sistema basado en los requerimientos anteriormente listados. En dicha Figura, se puede visualizar que el usuario invitado es el único actor que no tiene forma de iniciar sesión o registrarse en el sistema, pero si tiene la capacidad de poder interactuar con este, a través de proveer datos personales.

Los demás actores si tienen la capacidad de poder iniciar sesión. Cada una de las acciones que realizan los actores son las especificaciones que han sido dadas en los requerimientos. Lo más importante en este diagrama es la identificación de dichos actores.

En resumen, la lista de actores del sistema es:

- Administrador del sistema: Es el administrador general del sistema y es quién tiene todos los permisos para poder gestionar información en el sistema.
- Usuario registrado: este usuario tiene la capacidad de crear equipos y enviar la invitación a otros participantes a formar parte del equipo creado para poder participar en un torneo.

- Usuario invitado: este usuario cumple el rol de miembro de un equipo al cual ha sido invitado, previa confirmación.
- Usuario visitante: este usuario únicamente es un usuario que navega sobre la web y las actividades que puede desarrollar están limitadas a ver el ranking y si lo desea, registrarse en el sistema.

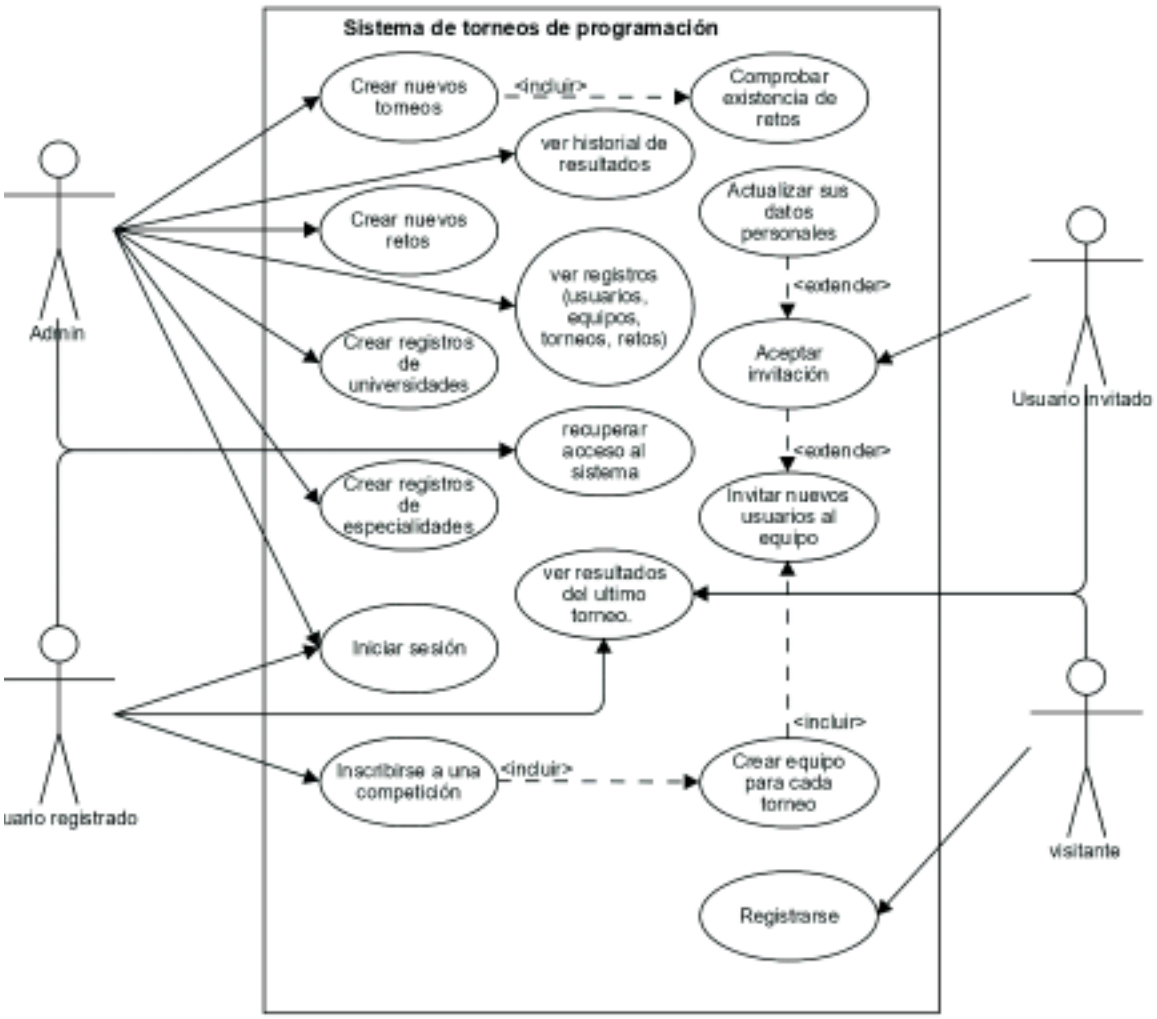


Figura 2.13. Diagrama de caso de uso – sistema de torneos de programación

2.2.2. Diagramas de secuencia

Ahora, es importante definir el ciclo de las actividades más importantes que se van a ejecutar sobre el sistema utilizando diagramas de secuencia.

Esta secuencia se obtiene, nuevamente, basado en los requerimientos funcionales especificados anteriormente. Esto permitirá tener una mejor comprensión de cómo debe funcionar el sistema en diferentes casos.

Es importante mencionar que dichos diagramas se acercan lo mayor posible al flujo real para implementarlo en el prototipo.

Para el registro de usuarios se debe tener un mayor control en cuanto a validación de datos; el flujo será como se muestra en la Figura 2.14. El mismo ciclo de registro de usuarios se utiliza para realizar el proceso de recuperación de la contraseña.

Se ha considerado utilizar un *token* de recuperación en la URL y está, es adjuntada al mensaje de correo que será enviado al usuario, para poder cambiar la contraseña.

Esta consideración viene dada por el hecho de que el mecanismo de recuperación se lo hace informando al usuario a través del envío de un correo electrónico a la cuenta personal de dicho usuario.

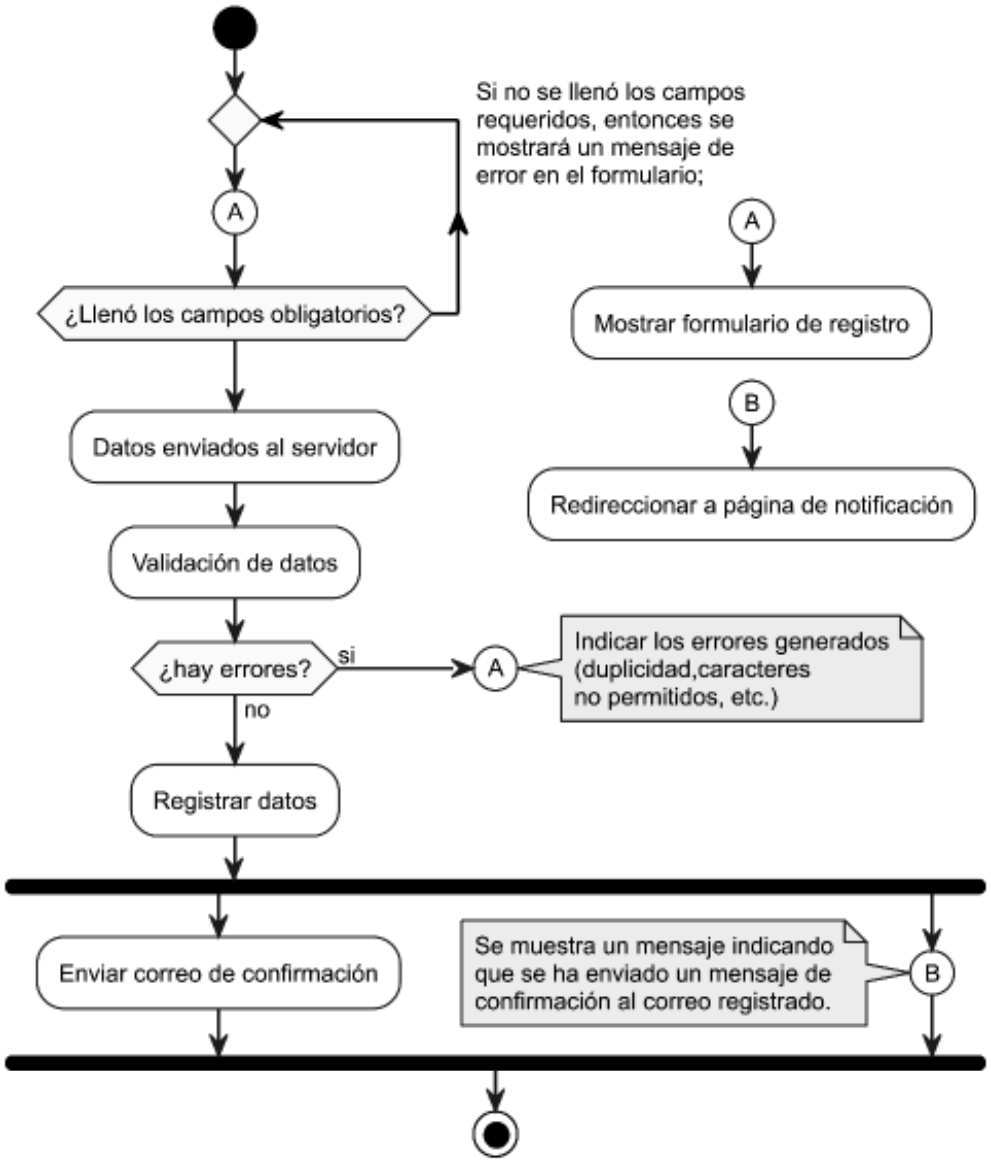


Figura 2.14. Ciclo de registro de usuarios

Dado que existen registros y actualizaciones de datos que siguen un mismo flujo, en la Figura 2.15, se presenta un diagrama UML de secuencia que representa dicho flujo común. Este diagrama cubrirá de forma general todos aquellos ciclos de registro y actualización de datos en el sistema.

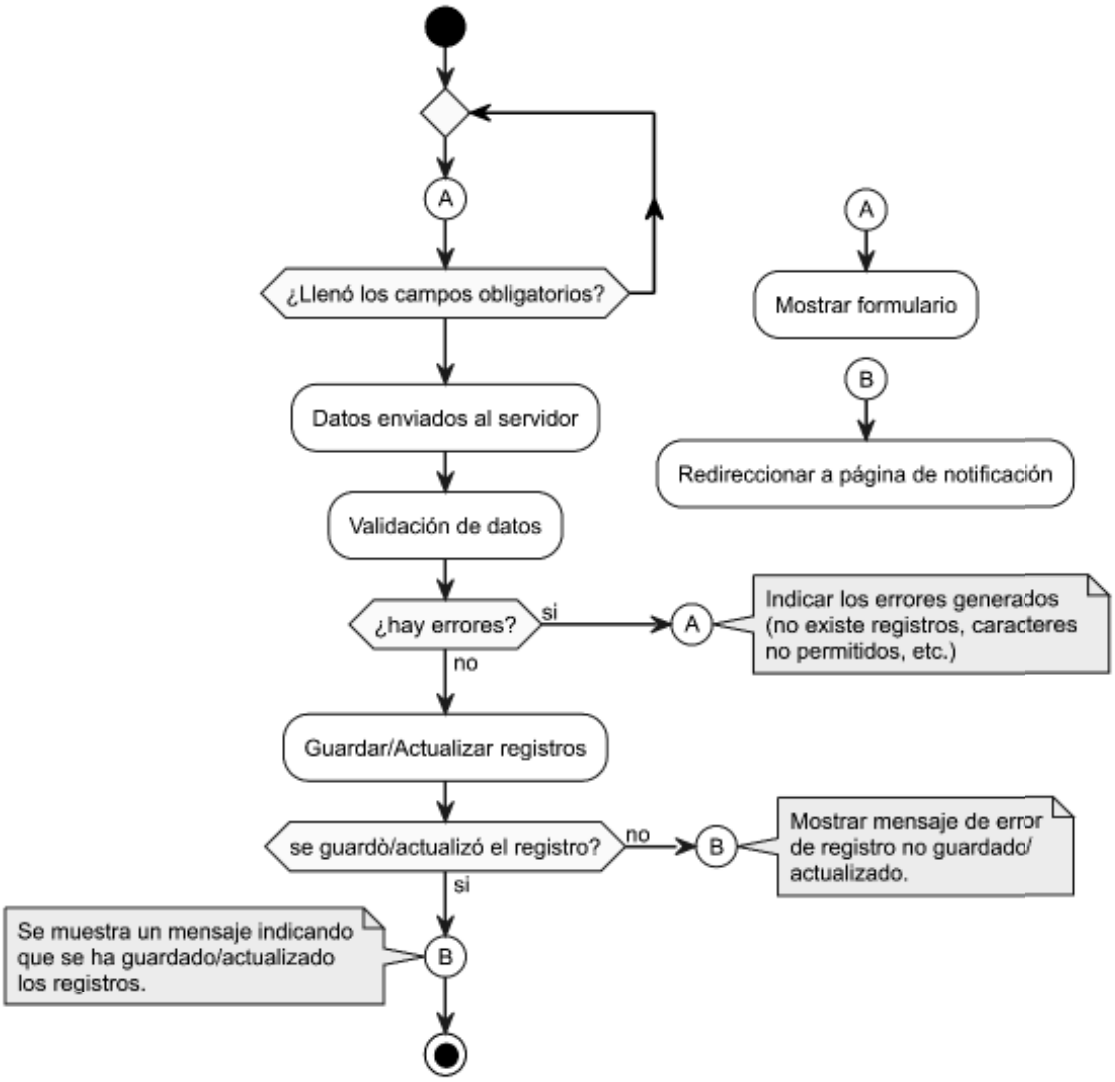


Figura 2.15. Ciclo general de registro o actualización de información

Como se puede visualizar en la Figura 2.16, uno de los requerimientos es que, para crear un torneo, previamente debe existir un registro de ejercicios propuestos. Esta condición viene dada por el hecho de que los ejercicios deben ser reutilizables para cualquier torneo que se vaya a crear.

En caso de que esta condición no se cumpla, el usuario deberá ser notificado con un mensaje de error el cual indique que es necesario crear ejercicios propuestos antes de poder crear un torneo. Además, para la creación de los torneos, debe existir la condición

de que se seleccione al menos un problema planteado y un máximo de 5 problemas que se puedan resolver en un torneo de programación. Esto con propósitos demostrativos. Sin embargo, dicha limitación no debería ser obligatoria, sino que pueda ser configurable y que sea el administrador del sistema quién determine cuantos ejercicios como máximo y como mínimo, se puedan especificar en un torneo de programación.

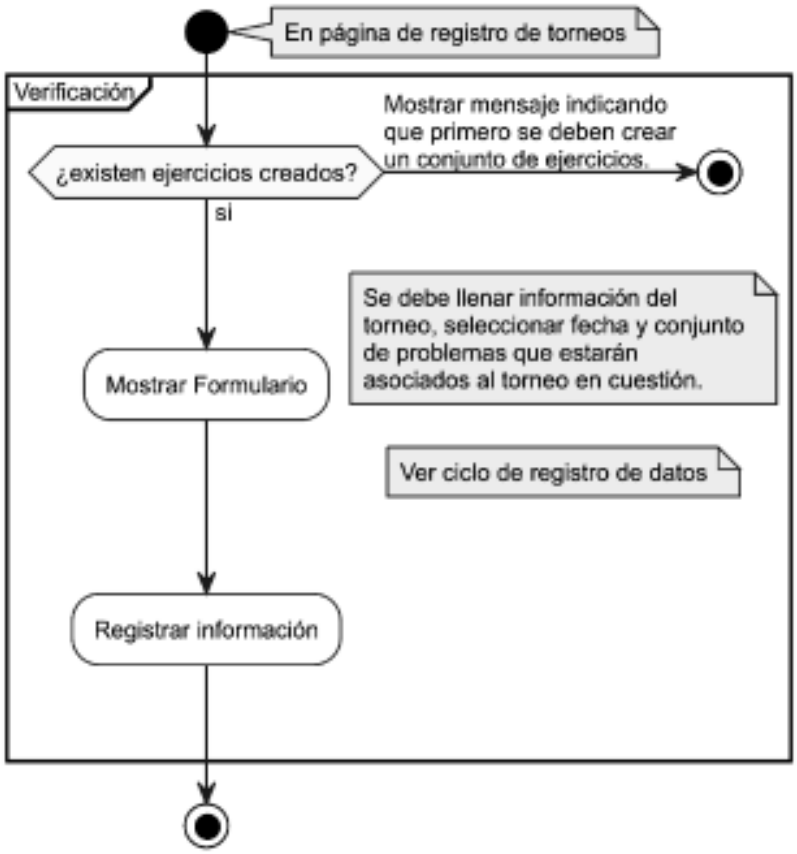


Figura 2.16. Ciclo de creación de un torneo

El diagrama de secuencia de la Figura 2.17, muestra el flujo de la ejecución de un torneo de programación.

Uno de los requerimientos dados en la lista de requerimientos que se han especificado anteriormente, es que, el equipo participante, haciendo uso del navegador web, puede verificar cada uno de los ejercicios que son propuestos y resolver el que más le convenga. Es decir, que se debe dar la libertad al participante de ver por cual ejercicio empieza a plantear la solución y, mientras esto sucede, el navegador no debe bloquearse incluso, cuando el resultado propuesto es enviado para ser evaluado.

Este detalle es muy importante tener en cuenta puesto que, al codificar el módulo para solventar el requerimiento, debe evitarse programar con procesos bloqueantes y, por

ende, evitar que el usuario tenga que esperar por la respuesta de la evaluación de un ejercicio, para poder continuar con el siguiente.

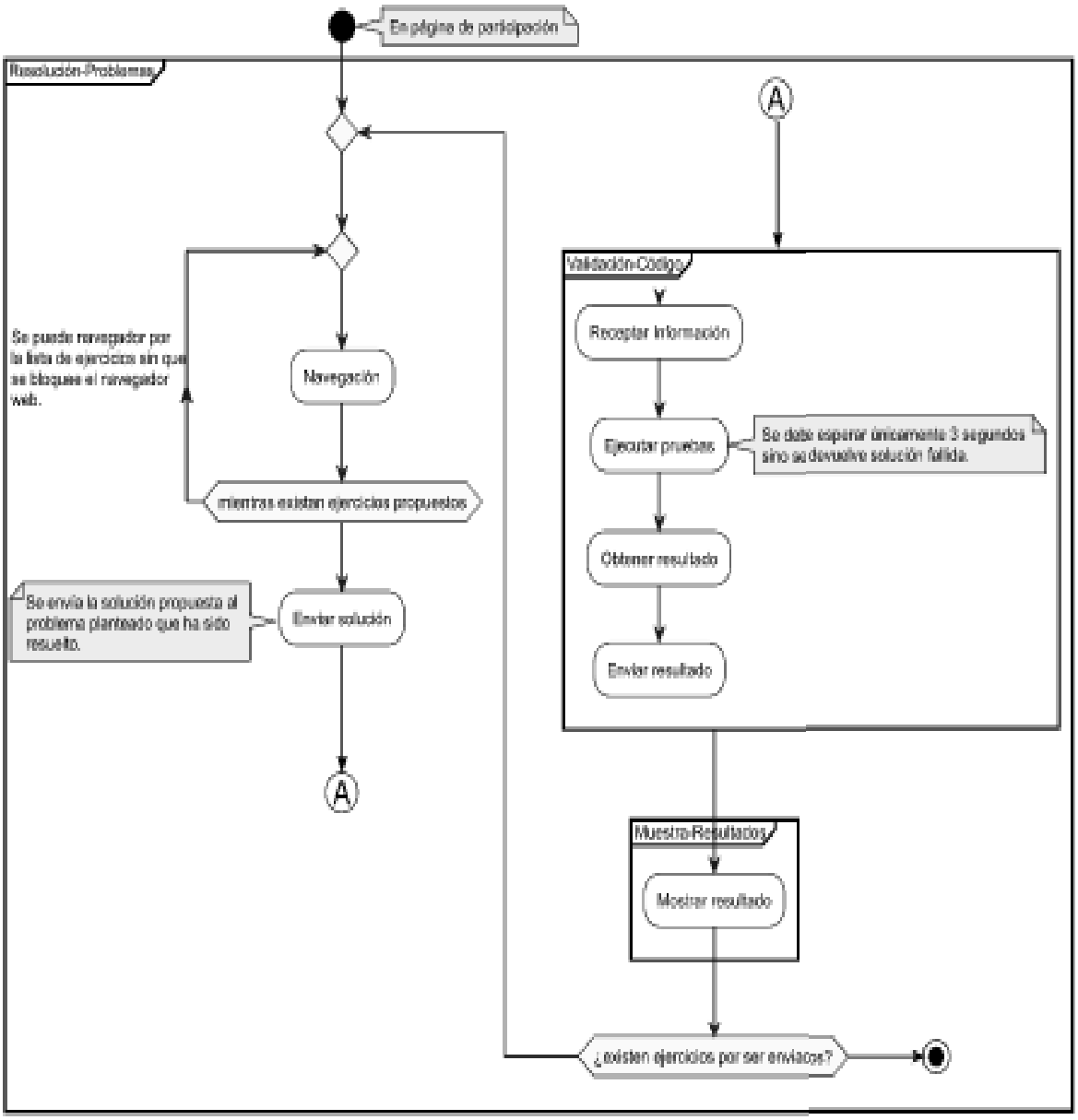


Figura 2.17. Ciclo de ejecución de un torneo

El diagrama de la Figura 2.18, indica el flujo de creación de un equipo que participará en el próximo torneo programado. Para crear un equipo se tienen las siguientes condiciones:

- Debe existir un torneo próximo a llevarse a cabo.
- No debe estar en proceso un torneo.
- El usuario no debe tener un equipo formado. En este punto es importante que los equipos solo existan mientras el torneo dura.

- Los usuarios podrán formar un equipo nuevo para cada torneo.
- No se debe permitir crear un nuevo equipo hasta que haya finalizado el torneo que esté actualmente llevándose a cabo.

Validada las condiciones impuestas, el usuario ya podrá crear un nuevo equipo. En el proceso de creación del equipo el usuario podrá invitar a varios usuarios. El límite de usuarios se define en la creación de nuevos torneos.

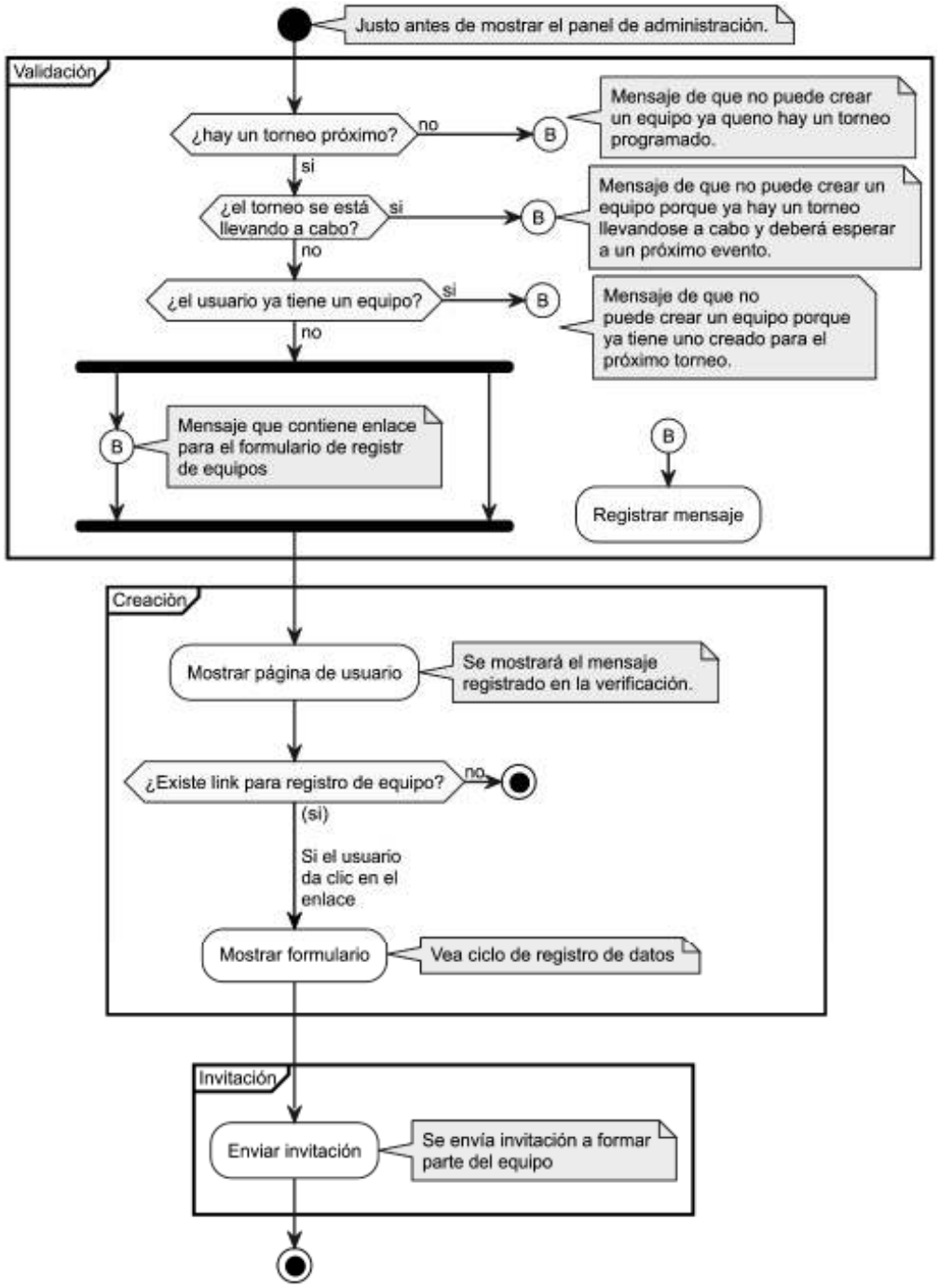


Figura 2.18. Flujo de creación de un equipo

También es importante precisar el ciclo de confirmación de usuarios a unirse a un equipo previa invitación. La Figura 2.19, muestra el flujo de confirmación. El usuario invitado deberá llenar información personal para pasar a formar parte de un equipo. Sin embargo, hay una consideración que se ha tomado en cuenta previo a llenar los datos solicitados. Esta consideración viene dada por el hecho de proveer un mecanismo de confirmación basado en un identificador único para cada invitado. Este identificador será utilizado para verificar que dicho usuario ha sido invitado a formar parte de un equipo. Si el identificador es modificado, dicho usuario no podrá acceder al formulario de registro de confirmación.

Cuando el usuario haya llenado la información correspondiente, ya podrá aparecer en la lista de participantes del equipo en cuestión.

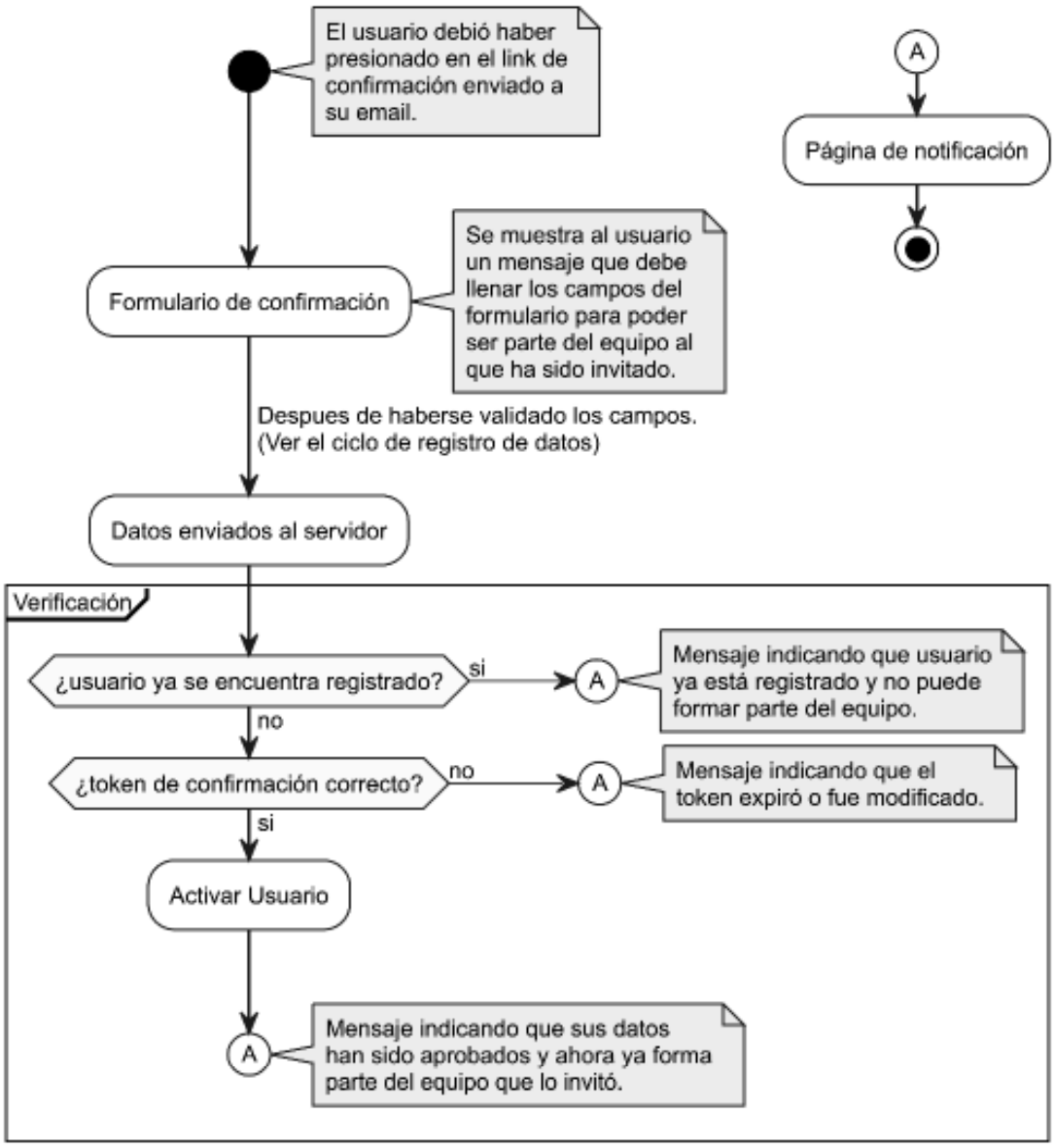


Figura 2.19. Ciclo de confirmación de participantes previa invitación

2.2.3. Diagrama Relacional de la base de datos

El diagrama relacional mostrado en la Figura 2.20, representa el modelo de la base de datos que actualmente se está utilizando en el prototipo. El diagrama relacional final se ha obtenido a través de los requerimientos planteados y en un ciclo iterativo en el proceso de implementación del prototipo. Varias tablas se relacionan con los actores identificados y los requerimientos del prototipo que han sido indicados de forma explícita. Otras tablas como `DataType`, `Level`, `Code`, `Score`, se han tenido que inferir de los requerimientos.

`User`: Corresponde a la tabla que almacena todos los detalles del usuario como datos personales, tipo de usuario, *tokens* que son adjuntados a la URL ya sea para verificar un usuario registrado, recuperar contraseña o confirmar la participación de un torneo.

`Carrier`: En esta tabla se guardan registros de las carreras universitarias. Hay una única columna que es actualizada por el administrador del sistema y corresponde al nombre de la carrera.

`University`: Mantiene información de las universidades. En esta tabla se registra el nombre de la universidad y si se desea, un acrónimo asociado a la universidad que se registre.

`Code`: Esta tabla se utiliza para registrar todos los casos de prueba asociados a un problema que es registrado en la tabla `ChallengingProblem`.

`Team`: Mantiene los registros de los equipos que van a participar en un torneo en particular. Adicionalmente, mantiene los identificadores que permiten determinar que lenguaje de programación utilizará el equipo y los resultados de cada problema que vaya resolviendo.

`TeamMembers`: Permite mantener una relación entre un usuario y el equipo al cual pertenece dicho usuario. Esta tabla solo registrará datos de aquellos usuarios que han confirmado ser parte del equipo al que fueron invitados.

`TeamTournament`: Esta tabla permite mantener una relación del equipo participante con el torneo que se esté ejecutando.

`Schedule`: Esta tabla permite registrar la fecha en que se llevará a cabo el torneo y también la hora de inicio y hora de finalización del torneo.

`ChallengingProblem`: En esta tabla, se registrarán todos los problemas propuestos.

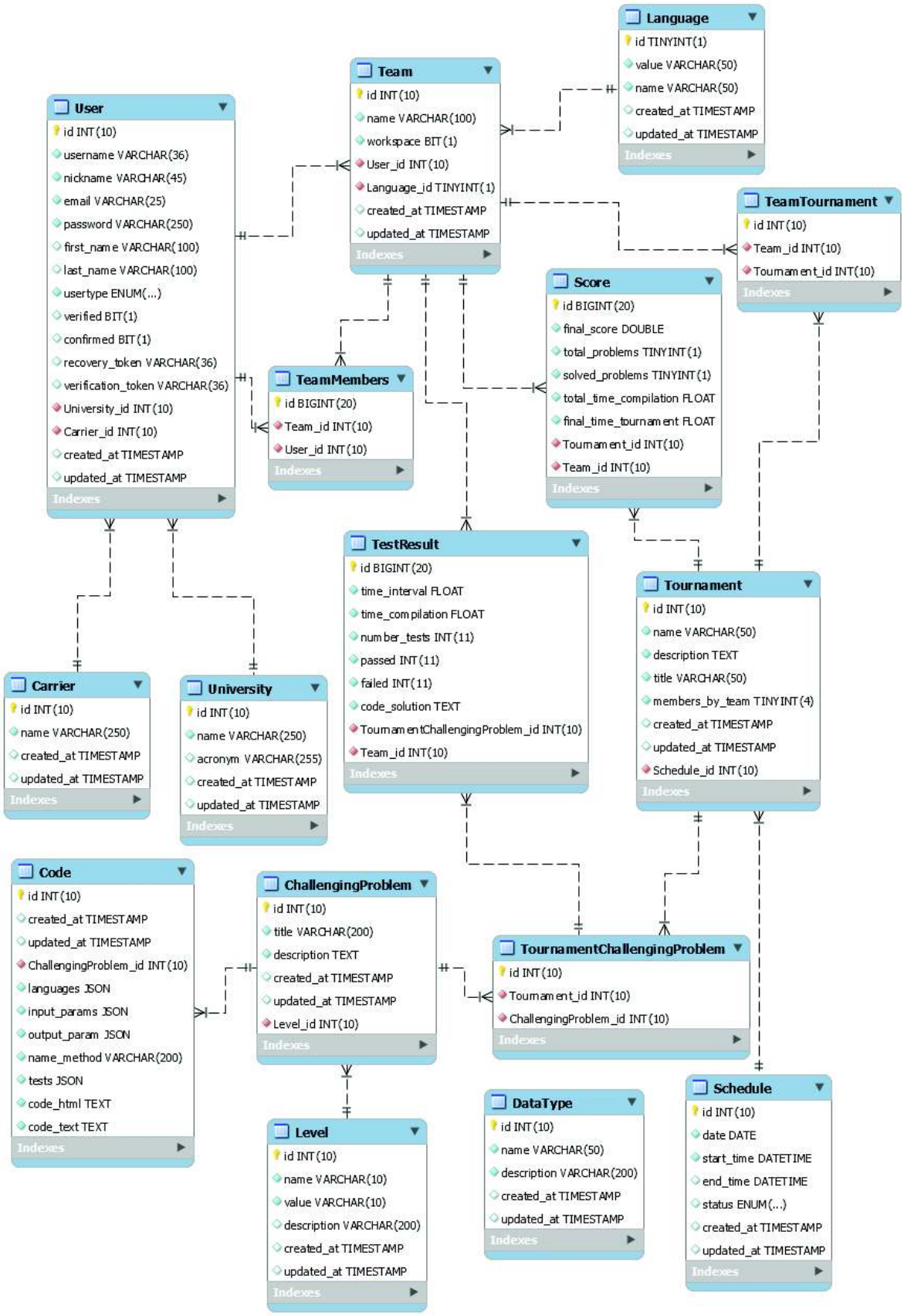


Figura 2.20. Diagrama relacional de la base de datos del prototipo

`Level`: En esta tabla se mantiene información que permitirá categorizar los problemas en tres niveles requeridos. Estos son: Básico, Medio, Difícil.

`Tournament`: En esta tabla, se almacenan datos de los torneos. Estos datos corresponden a información básica del torneo como el nombre y la descripción, así como el número de participantes que son permitidos para cada torneo.

`TestResult`: En esta tabla se registran los resultados parciales de cada problema que va resolviendo un equipo. Estos datos serán utilizados para calcular el resultado final.

`Score`: En esta tabla se registra el resultado final del cálculo de los valores parciales registrados en la tabla `TestResult`, cuando se haya finalizado el torneo.

`Languaje`: En esta tabla se mantiene información de los dos lenguajes que soporta el prototipo. Contiene una columna `value` que corresponde a un identificador que es utilizado por el sistema para crear las clases para los casos de prueba y la columna `name` que corresponde al nombre del lenguaje de programación.

`TournamentChalleningProblem`: Esta tabla permite guardar los identificadores de las tablas `Tournament` y `ChallingProblem` con el propósito de disponer de un registro entre los problemas planteados y el torneo que se está llevando a cabo. El resultado de esta asociación permite generar un identificador que será pasado a la tabla `TestResult` cada vez que se vaya resolviendo un problema. Así, de esta manera, se podrá identificar fácilmente que problemas están asociados a un torneo y si estos ya han sido evaluados para no presentarlos nuevamente si el usuario refresca el navegador.

`DataType`: Es una tabla que mantiene registros de los tipos de datos que son soportados por el sistema. Los datos que son soportados por el sistema son: `Short, Chart, Int, Long, Float, Double` y `String`.

2.2.4. Wireframes

A continuación, se muestra una serie de figuras que serán referentes para poder implementar la interfaz gráfica del sistema final. Estos diseños como tal, ayudan a tener una mejor compresión de la estructura que se desea tener en la aplicación web. La Figura 2.21, representa a la página principal del sistema. Este espacio está enfocado en mostrar el tiempo que falta para el próximo torneo e incluso, un enlace al sistema de *ranking* del último torneo llevado a cabo. En la sección inferior, un espacio para mostrar las tecnologías que han sido utilizadas para implementar el prototipo y finalmente, en la parte final inferior, el píe de página con información básica.

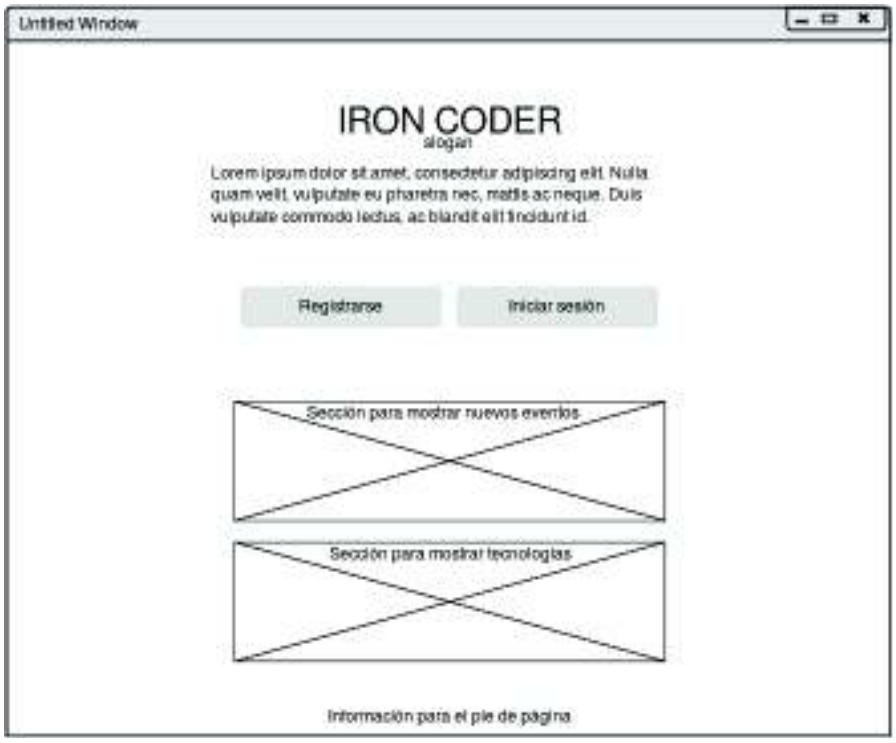


Figura 2.21. Página de inicio

Las Figura 2.22, Figura **2.23** y Figura **2.24** corresponden a las páginas de inicio de sesión, registro de usuarios y recuperación de contraseñas, respectivamente. En cada una de ellas se agrega enlaces de acceso rápido para cada página mostrada anteriormente.

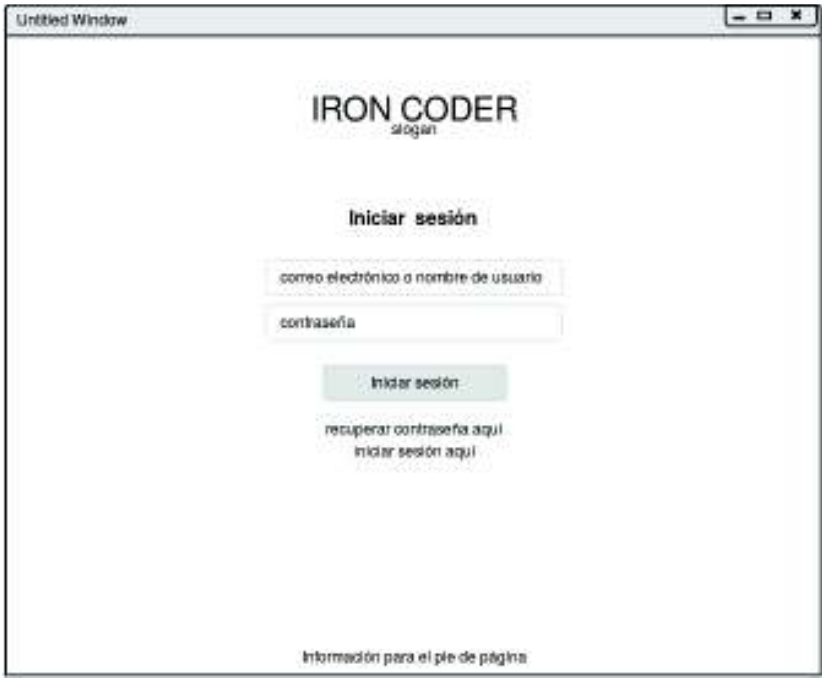


Figura 2.22. Inicio de sesión

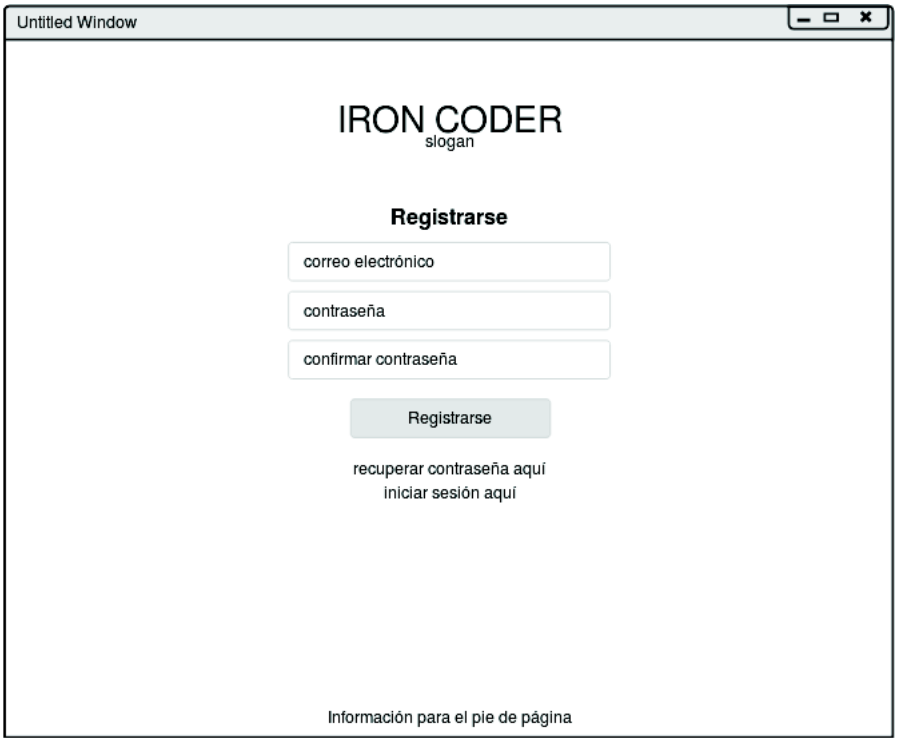


Figura 2.23. Registro de usuarios

Figura 2.24. Recuperación de contraseña

La Figura 2.25, corresponde al área que será utilizada para las notificaciones cuando el usuario haya sido redireccionado, o incluso cuando se intente ingresar a un área restringida como por ejemplo al área de administración de sistema.

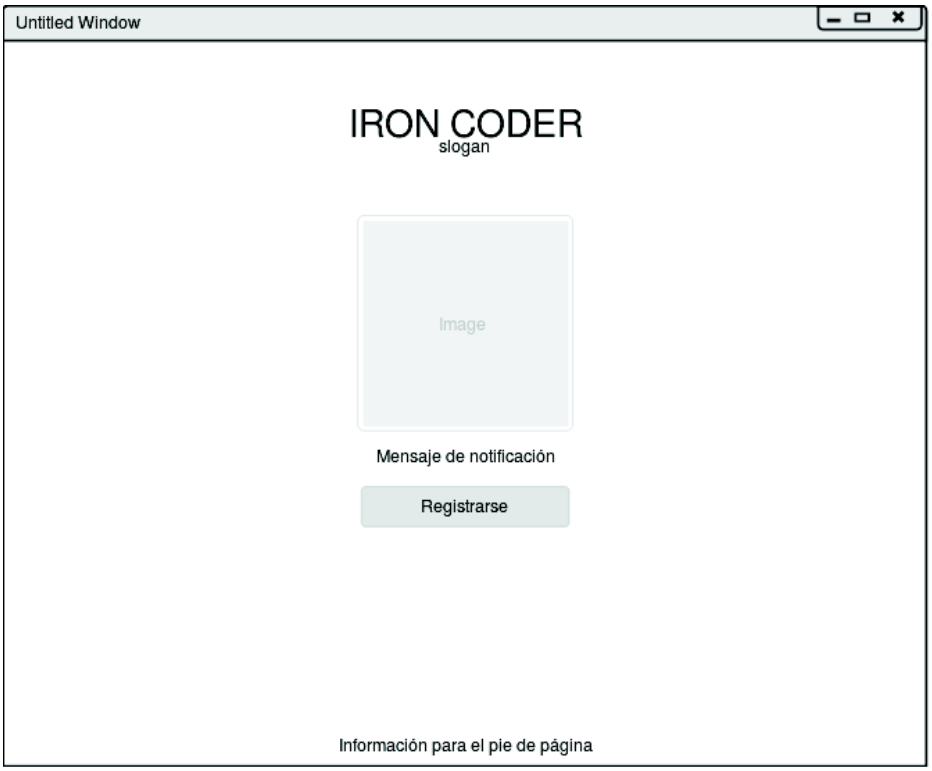


Figura 2.25. Área de notificaciones

La Figura 2.26, muestra de forma general cómo será el formulario utilizado para actualizar información propia del usuario.

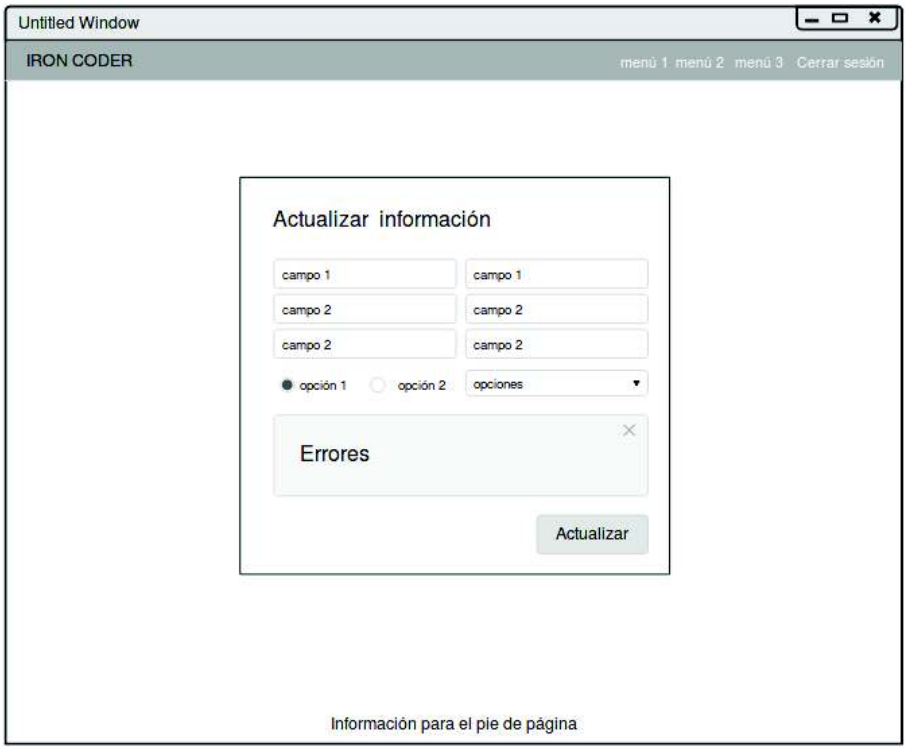


Figura 2.26. *Wireframe* común para actualizar datos de usuario

Las Figura 2.27, Figura **2.28**, Figura **2.29** y Figura **2.30**, presentan *wireframes* que serán utilizados en la sección de administración del sistema. En primera instancia, se muestra en la Figura 2.27, el *wireframe* del panel de administración el cual contiene varias secciones utilizadas como accesos directos a crear contenidos y ver listas de contenidos. Para acceder a ellos, se provee un botón en la esquina superior derecha de cada bloque para que este, despliegue las opciones para crear contenidos y ver listas de contenidos.

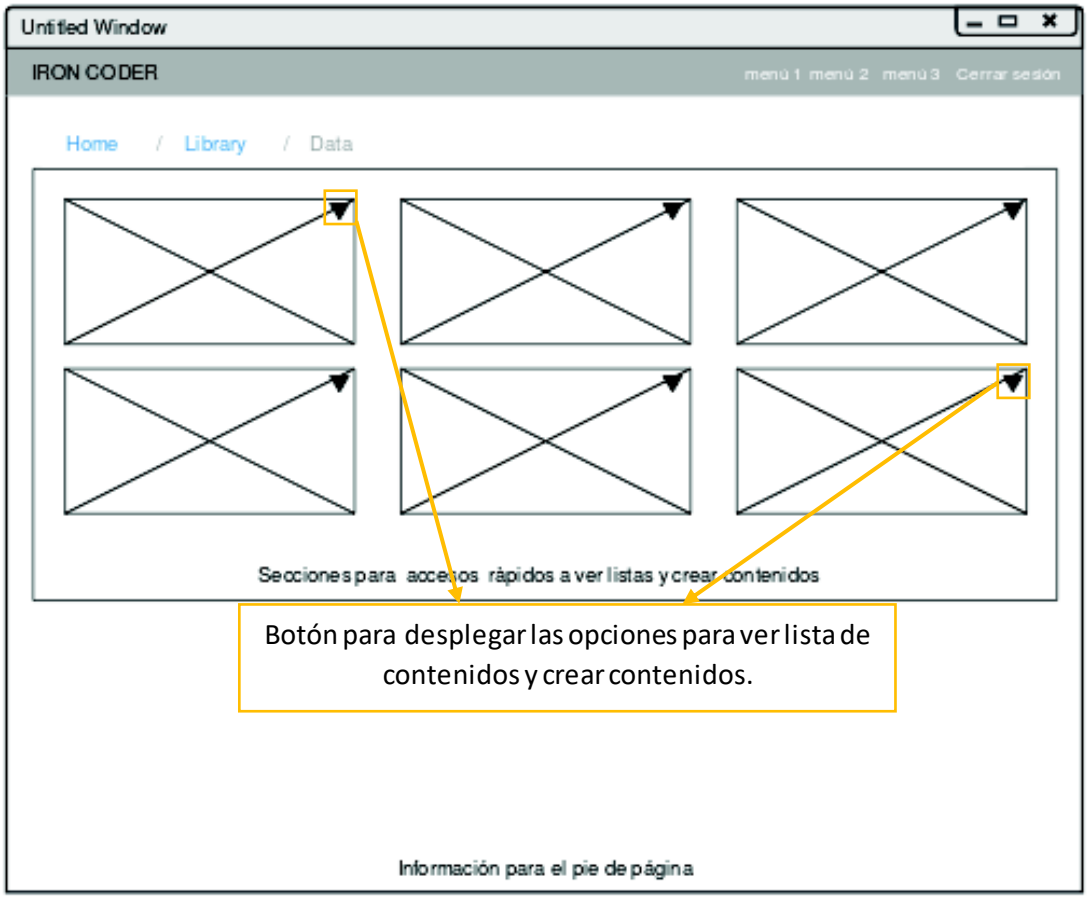


Figura 2.27. Panel de administración

La Figura 2.28, muestra de manera general, cómo será el espacio para ver listas de contenidos. Como se puede visualizar, existe una tabla con el contenido a ser mostrado y en la parte superior derecha una barra de búsqueda para obtener cualquier información.

La Figura 2.29 representa el diseño de los espacios para crear problemas propuestos y crear torneos.

En la Figura 2.30, se puede visualizar cómo será el esquema de *ranking* de los torneos realizados; contiene un espacio que resalta a los 3 primeros participantes y una lista para presentar a los demás participantes.

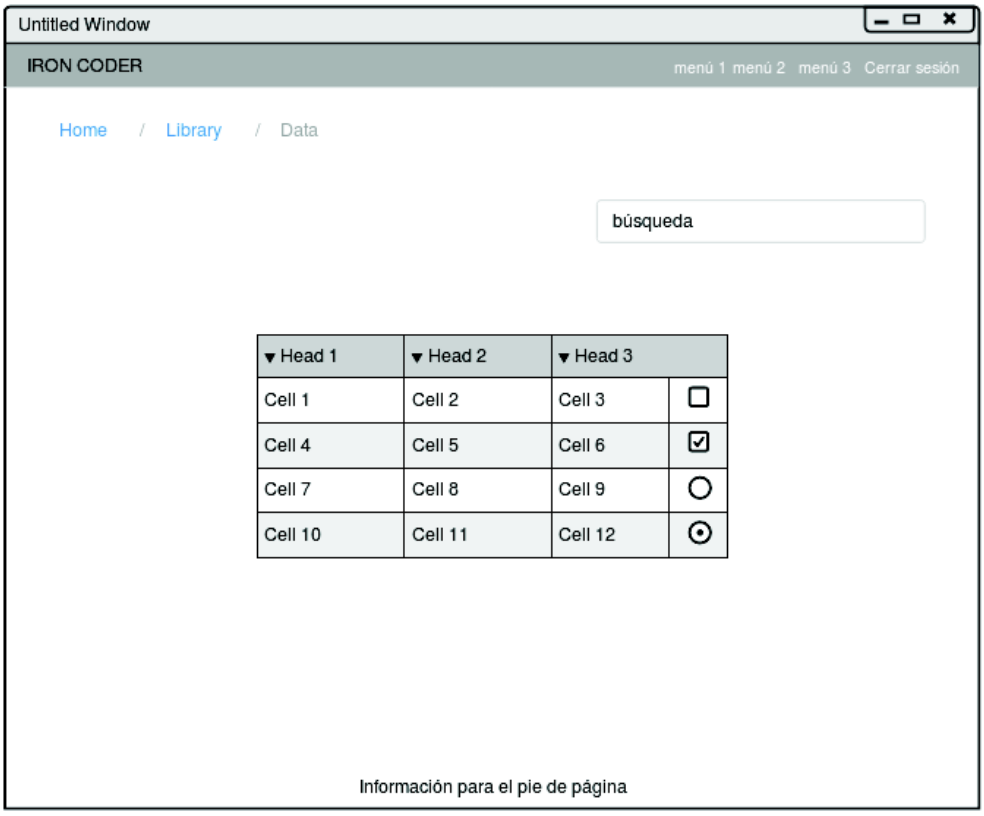


Figura 2.28. Listas de contenidos

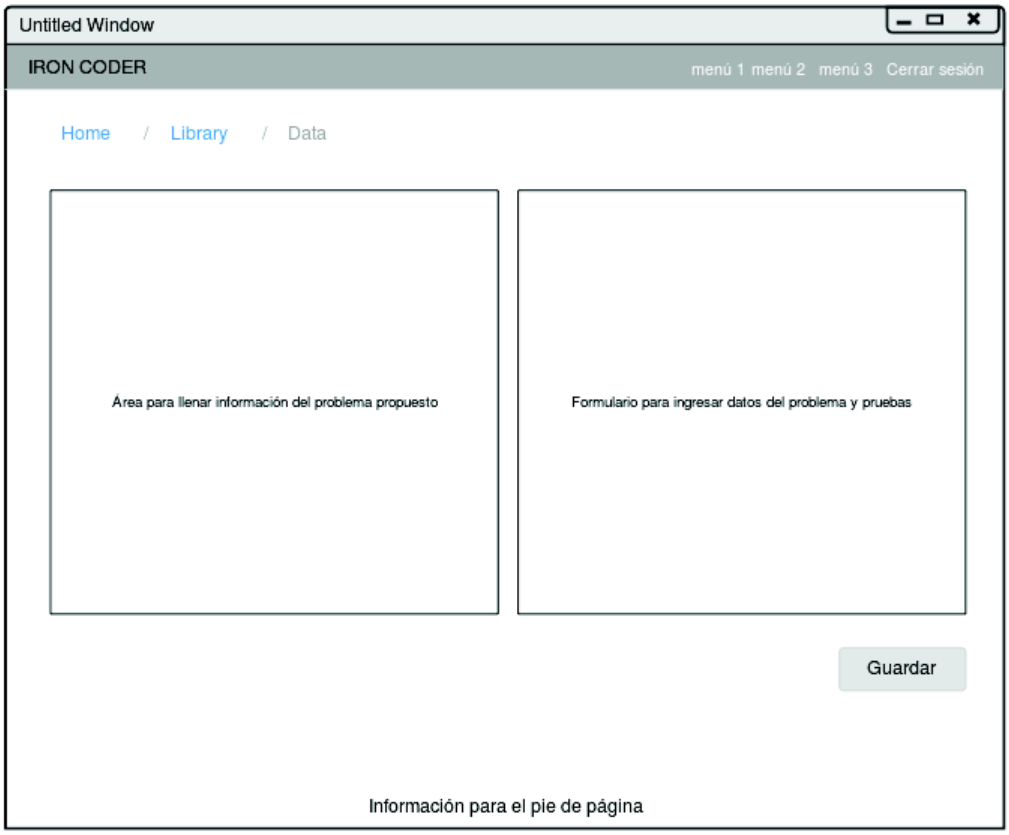


Figura 2.29. Crear problemas / torneos

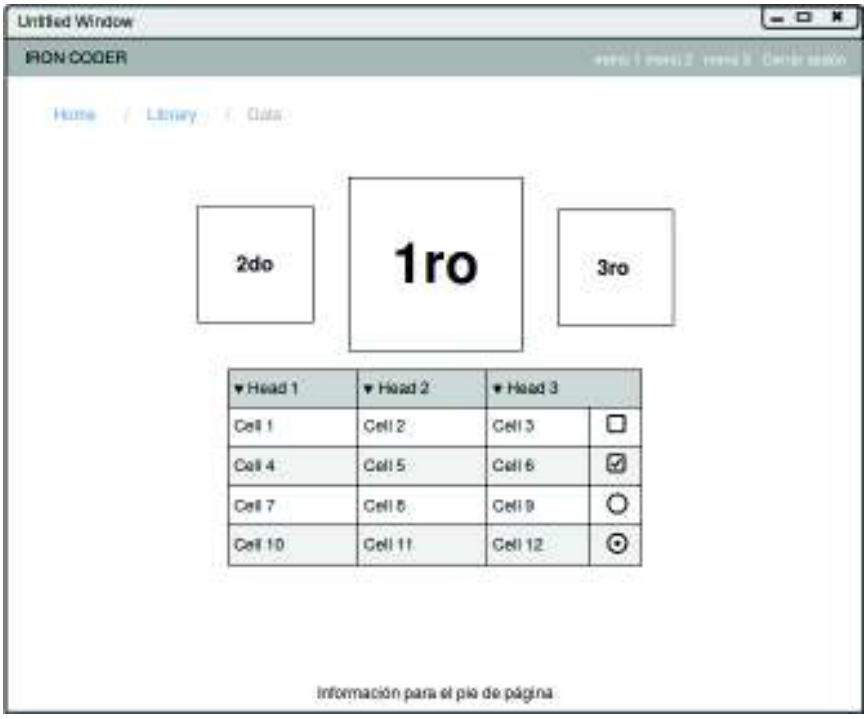


Figura 2.30. *Ranking* de competidores del último torneo

La Figura 2.31, representa a la página que verá el usuario cuando éste haya iniciado sesión. En esta sección se tiene dos bloques que corresponden a un área de notificaciones y un área para informar de un próximo torneo o, si ya se está llevando a cabo un torneo, proveer un enlace de acceso de dicho torneo.

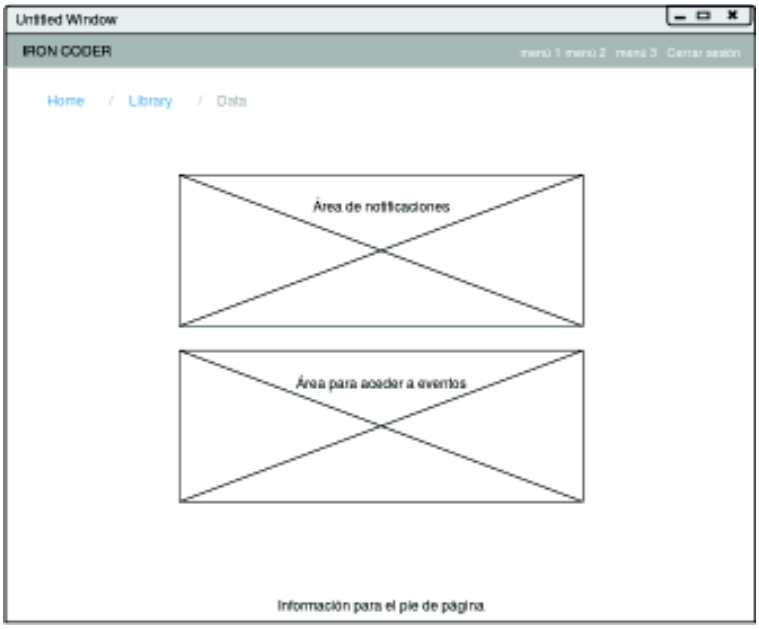


Figura 2.31. Página que verá el usuario cuando haya iniciado sesión

En la Figura 2.32, se puede ver el área de competición que está compuesta por un espacio para la información del problema, un editor de texto, una sección para navegar por los problemas planteados, un espacio que muestra el tiempo restante del torneo y una sección para mostrar los resultados de evaluación.

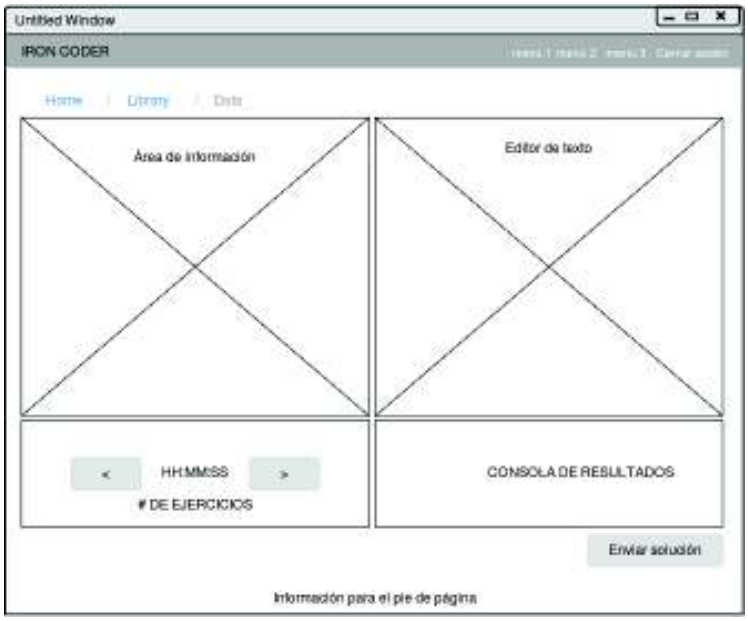


Figura 2.32. Área del editor de código

2.2.5. Arquitectura del Sistema

Es importante tener claro cómo será la arquitectura del sistema. Esta arquitectura está determinada por las especificaciones del prototipo, así como los elementos que conformarán el mismo. En la Figura 2.33, se muestra de manera general la arquitectura del prototipo y los elementos que la conforman son: aplicación web (1), servicios web (2) y base de datos (3).

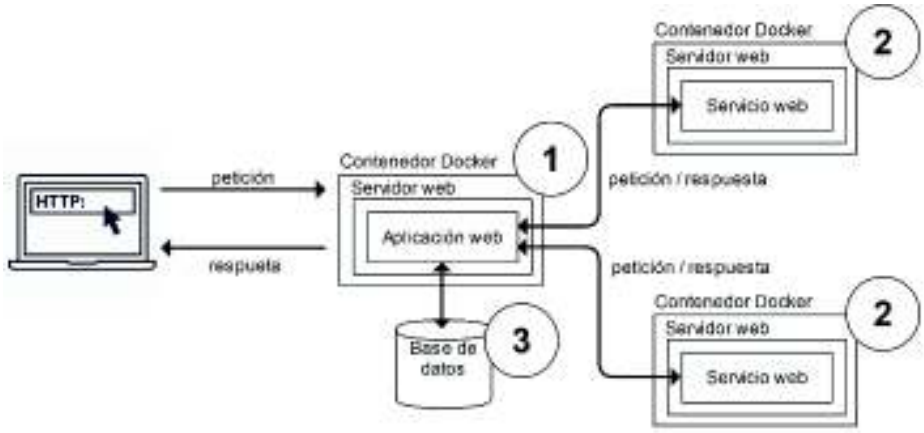


Figura 2.33. Arquitectura básica del prototipo

En la Figura 2.34, se muestra con mayor detalle la arquitectura del prototipo final y la interacción de los elementos más representativos que componen a dicha arquitectura.

Como se puede observar, dentro de la aplicación web (1), se muestra de forma general las capas MVC y la interacción que existe entre los elementos que forman parte de cada capa.

Cuando se realiza una petición a la aplicación web desde el navegador web, el enrutador de la aplicación web, maneja dicha petición y llama al controlador correspondiente y que forma parte de la capa Controlador.

Dicho controlador, delegará tareas a elementos que forman parte de la capa Modelo, para gestionar la información en la base de datos a través de la utilidad MySQL *driver* o, delegará tareas a elementos de la capa Vista, para generar una página HTML que pueden incluir componentes web, utilizando la utilidad *render*.

Además, el elemento *WebSocket* que forma parte de la capa Controlador, puede interactuar con el servicio web, a través de la utilidad HTTP *request*.

Esta interacción está asociada con procesos para crear espacios de trabajo, crear los archivos de prueba, etc.

El servicio web (2) que se muestra en la Figura 2.34, se encargará de realizar tareas de compilación, ejecución de pruebas, creación de espacios de trabajo para los equipos competidores, creación de archivos para pruebas, entre otros.

El enrutador del servicio web (2), se encargará de procesar las peticiones que son realizadas desde la aplicación web (1).

Elementos de la capa Controlador, delegarán tareas a la utilidad CLI, para que se encargue de realizar acciones cómo, por ejemplo, crear carpetas para cada equipo, archivos que contienen el código de pruebas, archivos que contienen los algoritmos que son enviados por los equipos participantes.

Así mismo, esta utilidad puede interactuar con archivos de *script* que ejecutan las tareas de compilación, ejecución y evaluación de código utilizando los *frameworks* de pruebas unitarias JUnit y NUnit.

El resultado de evaluación se devolverá al CLI y éste, delegará la tarea de dar el formato correcto a la información que contiene el resultado, para que se obtenga la estructura correcta en formato JSON para ser devuelta a la aplicación a través de la capa Controlador.

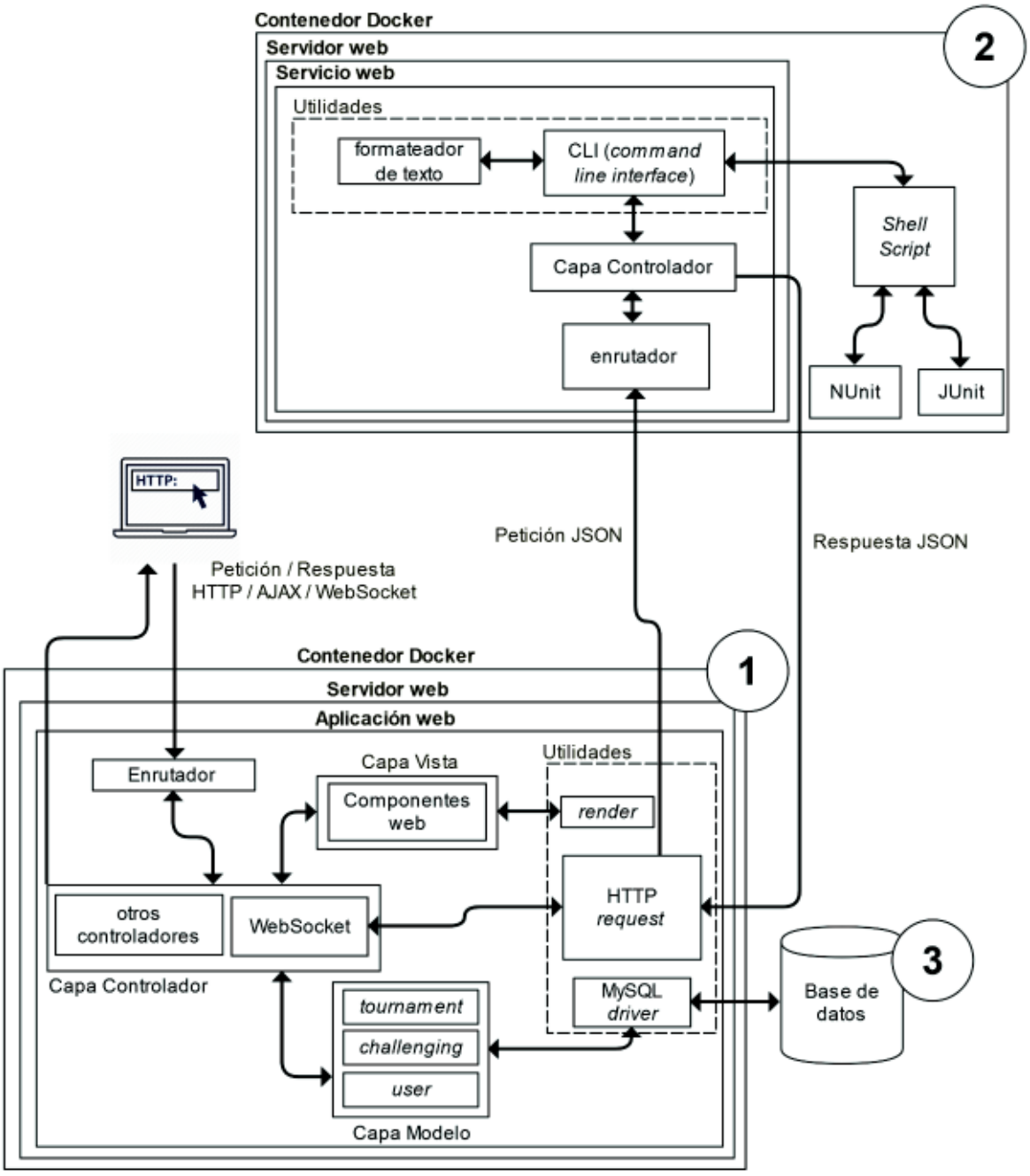


Figura 2.34. Arquitectura final del sistema en un entorno local

Como el despliegue del prototipo se lo realizará utilizando los servicios de *Amazon Web Service*, la arquitectura mostrada en la Figura 2.34, tiene varios cambios pues, ahora se deben considerar otros elementos que son parte de la infraestructura de AWS para proveer mecanismos de configuración para el despliegue tanto de la aplicación como de los servicios web y la base de datos, por lo que la arquitectura sobre la nube de AWS, se presenta en la Figura 2.35. Es importante mencionar que estos cambios son provistos automáticamente por AWS, cuando se crea algún recurso que se vaya a utilizar.

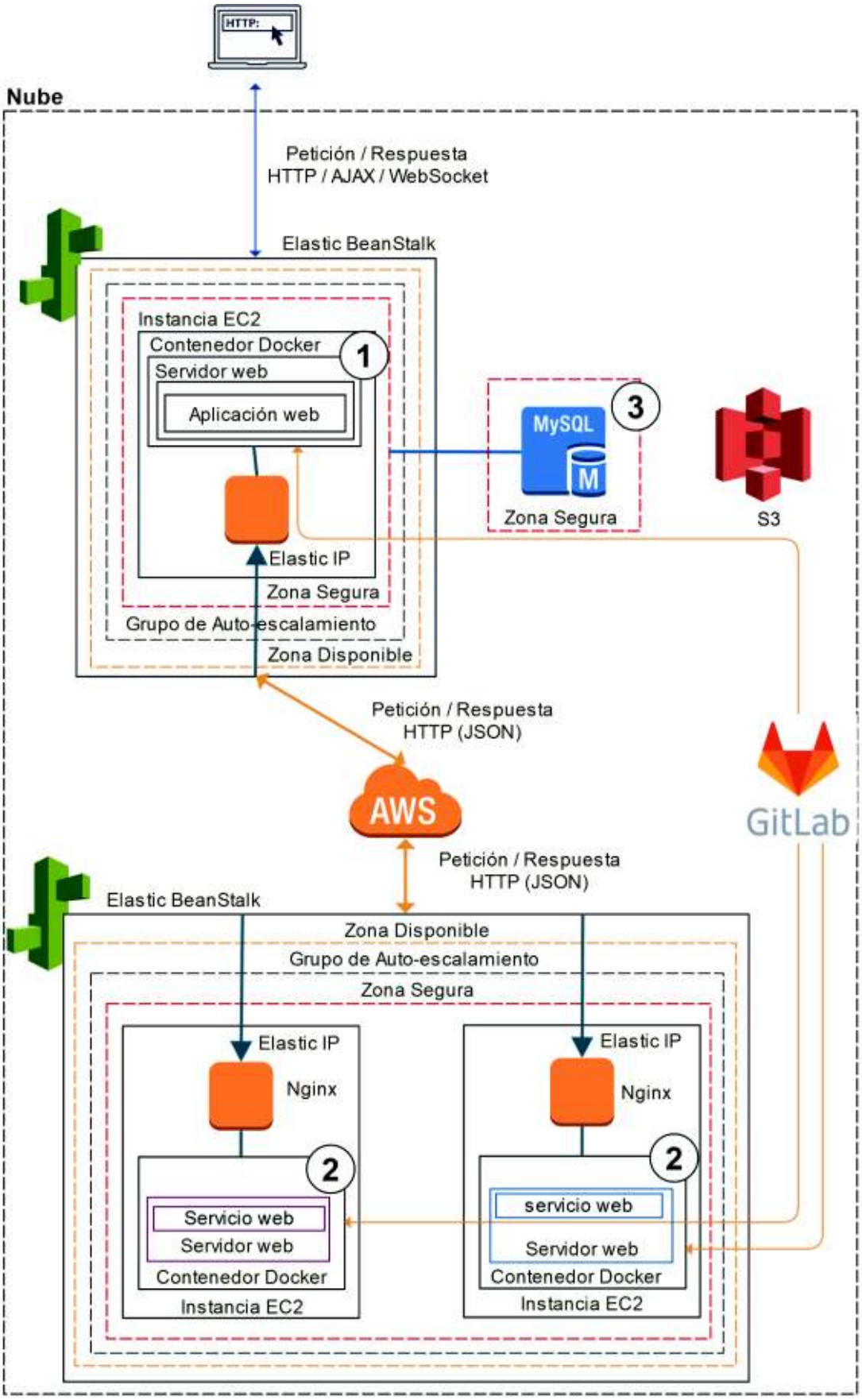


Figura 2.35. Arquitectura final del sistema en AWS

Como se puede visualizar en la Figura 2.35, cada uno de los elementos que conforman el prototipo están contenidos sobre varios entornos. Además, hay una serie de servicios adicionales que son utilizados en la puesta a producción del prototipo.

La zona disponible corresponde a la región donde existen los centros de datos de AWS y es donde serán desplegados los servicios.

El grupo de auto-escalamiento son grupos de instancias EC2 administradas por el servicio AWS *auto-scaling*. Se puede configurar el número de instancias a ser desplegadas. Para el presente proyecto, la configuración será de una única instancia pues los costos incrementan cuando el servicio de auto-escalamiento aumenta a dos o más instancias.

La zona segura corresponde a secciones lógicas en Linux que funciona como un *firewall* virtual. AWS crea las reglas necesarias para la seguridad de las instancias EC2 que se vayan a desplegar.

Cada instancia EC2 está conformada por un servidor Nginx que trabaja como un *proxy* para las peticiones entrantes y la instancia EC2. A este servidor se le asigna un nombre de dominio denominada ElasticIP. Este nombre de dominio tiene una estructura como se muestra en la Figura 2.36.

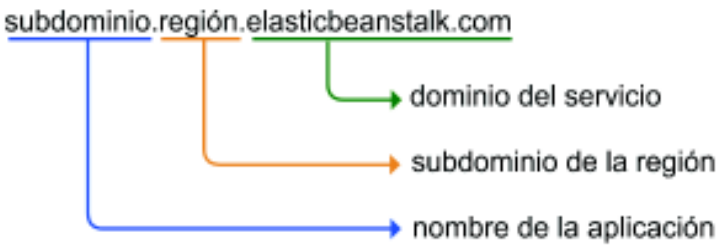


Figura 2.36. Estructura de una ElasticIP de AWS

Tanto S3, como GitLab, serán utilizados cuando se realice el despliegue de la aplicación pues en dichos servicios, estarán disponibles ciertos elementos que son importantes tanto para la configuración como para la instalación de los servicios y la aplicación web.

2.3 Implementación

En esta etapa también se definen una serie de actividades en un nuevo tablero Kanban (Figura 2.37). Se analizarán aquellas actividades que tienen mayor importancia pues estas, son las que realmente cumplen con el denominado modelo de negocio de la aplicación. Dicho modelo, no es más que la lógica que tiene el prototipo, esto es: ser una aplicación para torneos de programación.

Figura 2.37. Tablero Kanban - Definición de tareas

Considerando los elementos que forman parte del *To-Do* en el tablero Kanban de la Figura 2.37, se ha decidido empezar por el desarrollo de aquellos elementos más importantes del *Front End* y, posteriormente, los elementos más representativos del *Back End*. Por lo tanto, se procede a seleccionar aquella tarjeta cuyo requerimiento es la implementación de un módulo para proveer un espacio de trabajo para los torneos.

Antes de pasar a explicar algunos elementos que conforman el sistema, es necesario detallar la configuración del ambiente de desarrollo.

2.3.1. Configuración del entorno de desarrollo

Antes de realizar la implementación de los módulos que conformarán el sistema, es necesario preparar el entorno sobre el cual se empezará a codificar cada uno de dichos módulos. Dicha configuración incluye, las herramientas Docker, Nodejs, Golang, y todas las dependencias que se requieren para Nodejs y Golang, así como las imágenes base

que se utilizarán en la construcción de las imágenes finales para ser desplegadas en AWS.

En la Tabla 2.1 se muestran las características del equipo que será utilizado en el ambiente de desarrollo.

Tabla 2.1. Características del equipo para el ambiente de desarrollo

Distributor ID	Ubuntu x86_64
Description	Ubuntu 17.10
Ubuntu 17.10	artful
Process	Intel(R) Core(TM) i5-3330 CPU @ 3.00GHz
Memory	8Gb DDR3

Para instalar Docker, realice las siguientes acciones:

1. Descargar el paquete `.deb` de https://download.docker.com/linux/ubuntu/dists/
2. Ejecutar a través de la línea de comandos, el comando presentado en el Código 2.1.

```
1    $ sudo dpkg -i /path/to/package.deb
```

Código 2.1. Comando para instalar el paquete `.deb` de Docker

Para instalar Nodejs en el ambiente de desarrollo, se debe ejecutar las sentencias, a través de la línea de comandos, presentadas en el Código 2.2.

```
1    curl -sL https://deb.nodesource.com/setup_10.x |sudo -E bash -
2    sudo apt-get install -y nodejs
```

Código 2.2. Descargar e instalar Nodejs

Nodejs será utilizado para poder trabajar con la herramienta Vue-webpack-boilerplate. Además, permitirá construir los recursos estáticos que se generen con la herramienta antes mencionada.

Para instalar Go, se procede a ejecutar las líneas de código a través de la línea de comandos presentadas en el Código 2.3.

La línea 1, descargará el código fuente de Go. Luego, el comando de la línea 2, desempaquetará el archivo que ha sido descargado. La línea 3, moverá, el contenido que fue descomprimido, a la ruta `/usr/local`. A continuación, desde la línea 4 a la 6, se agregan las variables de entorno al archivo `.profile` del usuario actual. Las variables de entorno se deben especificar para que las sentencias de Go puedan identificar donde

están los binarios sin especificar la ubicación de dichos binarios. Finalmente, se ejecuta el comando `source` sobre el archivo `.profile`, para recargar las variables de entorno. Con esto ya está disponible Go para ser ejecutado.

```
sudo curl -O https://storage.googleapis.com/golang/go1.9.1.linux-amd64.tar.gz
sudo tar -xvf go1.9.1.linux-amd64.tar.gz
sudo mv go /usr/local
sudo echo "export PATH=$PATH:/usr/local/go/bin" >> ~/.profile
sudo echo "export GOROOT=$HOME/go" >> ~/.profile
sudo echo "export PATH=$PATH:$GOROOT/bin" >> ~/.profile
source ~/.profile
```

Código 2.3. Código para instalar Golang

Por defecto, Ubuntu ya viene pre-instalado con Git. Sin embargo, se requiere realizar una configuración de registro de credenciales que permitan que esta herramienta pueda llevar a cabo acciones con servicios externos como GitLab o AWS si así se lo requiere.

1. Se debe realizar la configuración de credenciales de identificación. Para ello se ejecutan los comandos presentados en el Código 2.4.

```
git config --global user.name "David Núñez"
git config --global user.email "davip@live.com"
```

Código 2.4. Comandos de configuración de credenciales en Git

Se debe genera una llave de autenticación para cargar el proyecto de desarrollo a los repositorios creados en GitLab. Además, esta llave será utilizada para desplegar los contenedores Docker en AWS. El comando que se utiliza desde la línea de comandos para generar la llave de autenticación se presenta en el Código 2.5.

```
ssh-keygen -t rsa -b 4096 -C "your_email@example.com"
```

Código 2.5. Generando llaves SSH

La llave generada se encuentra en la ruta `~/.ssh/id_rsa.public`. A continuación, se describen los pasos que se deben seguir para agregar la llave generada, en una cuenta de GitLab previamente creada.

Esto con el propósito de poder clonar los repositorios que han sido creados en GitLab para contener los componentes de proyecto. En la Figura 2.38, se muestran los pasos que se deben seguir.

1. Se da clic en la parte superior en el icono de usuario (1) y, a continuación, se selecciona la opción *Settings* (2). Se redireccionará a una nueva página.

2. En la parte inferior izquierda, se selecciona la opción SSH Keys (3). Se agrega la llave obtenida en la sección generación de llaves de seguridad (4).

El propósito de gestionar el proyecto a través de repositorios en GitLab es permitir que tanto el proyecto de la aplicación web como el proyecto del servicio web, sean descargados como dependencias en la construcción de las imágenes de Docker para ser desplegados en los servicios de AWS. Además, se deben realizar algunas configuraciones que permitirán que todo el proceso de despliegue sea un proceso automatizado a través de *scripts* de Linux conjuntamente con las etiquetas que se declaran en el archivo Dockerfile para construir la imagen que será desplegada.

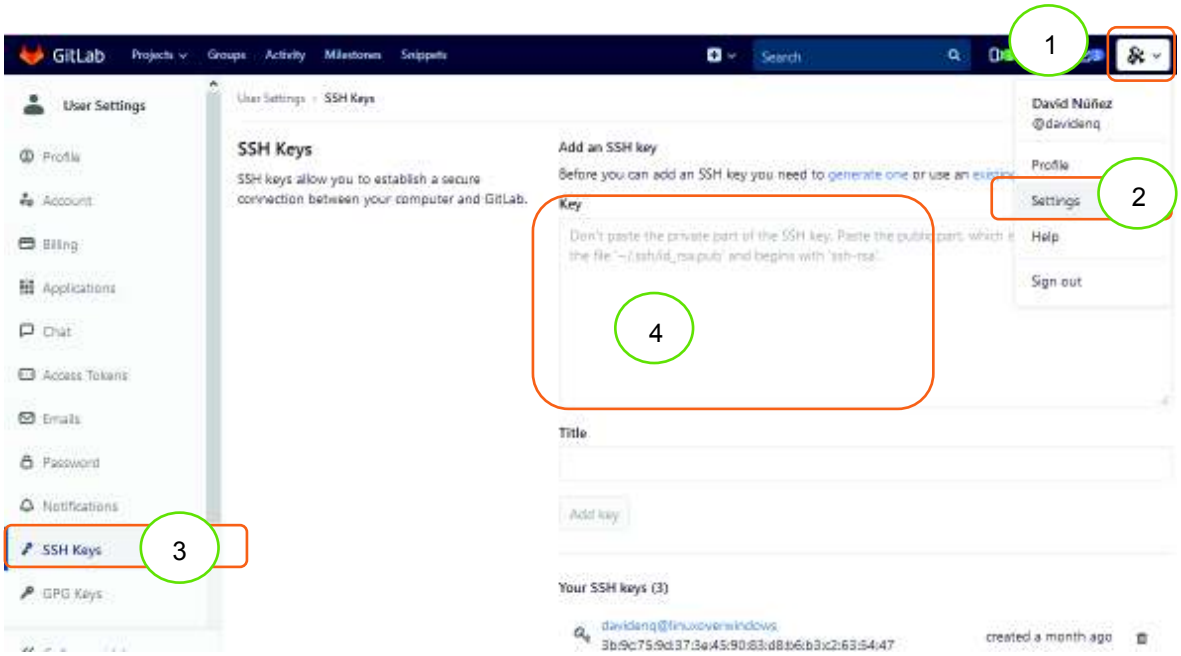


Figura 2.38. Sección SSH Keys en GitLab

En la Figura 2.39, se ve de forma gráfica la creación de un repositorio. Los pasos son:

1. Se da clic en la parte superior derecha (1). Se selecciona la opción *New Project* (2).
2. Se agrega un nombre que identifique al proyecto (3).
3. Se selecciona el nivel de visibilidad del proyecto *private* (4).
4. Finalmente se presiona el botón *Create Project* para crear el repositorio (5).

En la nueva página (Figura 2.40), se debe copiar la URL provista. Esta URL servirá para clonar el proyecto en el respositorio local y subirlo al repositorio remoto. Posteriormente, se clonará este repositorio en el proceso de creación de la imagen para los contenedores Docker.

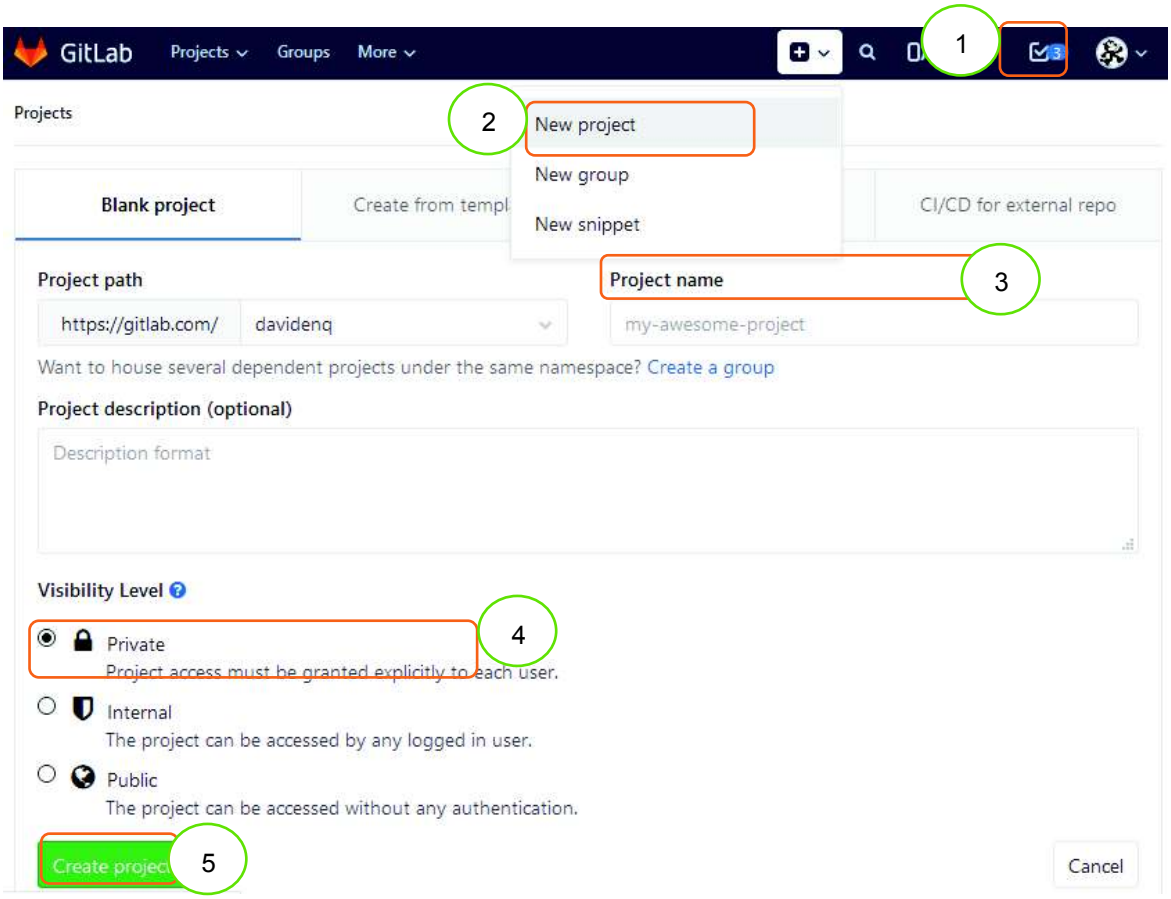


Figura 2.39. Crear nuevo repositorio en GitLab

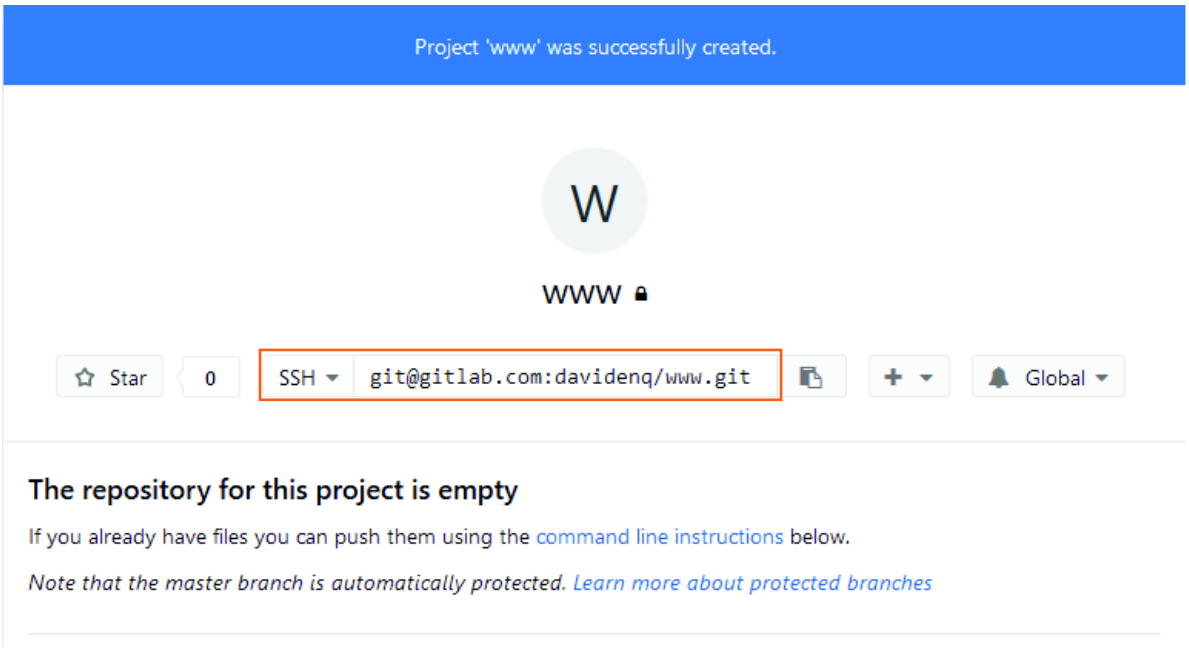


Figura 2.40. Repositorio creado en GitLab

El Código 2.6. Comandos para guardar cambios y subir a GitLab permitirá clonar el repositorio creado en GitLab y asociarlo con el repositorio que se encuentra en el

respositorio local. El comando de la línea 1, permitirá que Git sea utilizado en el proyecto actual.

El comando de la línea 2, clona el repositorio que previamente fue creado en GitLab, en el ambiente local. Los cambios que se van realizando sobre el código serán agregados a Git como el comando que se muestra en la línea 3. El comando de la línea 4 permite agregar un comentario que identifica que cambios se han realizado. Finalmente, el comando de la línea 5, subirá todos los cambios registrados en el repositorio local, al repositorio en remoto en GitLab.

```
1. $ git init
2. $ git remote add origin git@GitLab.com:davidenq/www.git
3. $ git add nombre_del_archivo_modificado
4. $ git commit -m "mensaje de los cambios realizados"
5. $ git push
```

Código 2.6. Comandos para guardar cambios y subir a GitLab

2.3.2. Aplicación web

En esta sección, primero se realiza una breve descripción de las carpetas que conforman la aplicación web. Posteriormente, se hace un breve análisis del contenido de cada carpeta y, finalmente, se realiza un análisis de los componentes web que han sido desarrollados para ser integrados en las páginas de la aplicación web.

a. Estructura de carpetas

En primera instancia, se ha modularizado la aplicación web como se ve en la Figura 2.41 y basado en el análisis realizado en la sección "Modularidad basado en el patrón de diseño MVC".

El archivo `main.go` es el punto de entrada al sistema. Aquí se gestionarán varias tareas asociadas con la carga de contenidos, verificación de disponibilidad de servicios web, carga de variables para los ambientes, entre otros.

En el archivo `routes.go` estarán todas las rutas de acceso al sistema. Estas rutas serán cargadas desde el archivo `main.go` cuando se haya inicializado la ejecución del servidor.

En el archivo `Gopkg.tom` estarán todas las dependencias requeridas por el sistema.

En el archivo `configuration.json` estarán variables de configuración que requieren algunas utilidades.

En el archivo `run.sh` estarán variables de entorno para la ejecución del sistema, así como el comando de ejecución del servidor.

Las variables serán utilizadas únicamente en el ambiente de pruebas, pues, para el ambiente de producción, AWS gestiona de otra manera dichas variables y ya no será necesario usar este *script*.

La carpeta `utils` contendrá utilidades que serán utilizados para tareas como conexión a la base de datos, generación de *tokens*, manejo de sesiones, *cookies*, entre otros.

En la carpeta `structs` se manejarán estructuras de datos de los modelos y de las peticiones y respuestas entre el cliente, la aplicación y el servicio web.

En la carpeta `static` se agregarán tanto los componentes web desarrollados, así como también las hojas de estilo, *scripts*, imágenes, entre otros, que son comunes para la aplicación web.

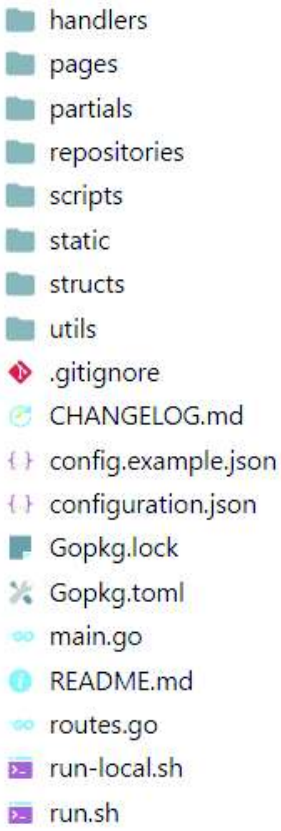


Figura 2.41. Modularización de la aplicación web

La carpeta `repositories` corresponde al modelo de MVC. Cada archivo está asociado con cada una de las tablas de la base de datos. En cada uno de ellos, habrá métodos para realizar consultas y actualizaciones de la información en la base de datos.

La carpeta `partials` contendrá plantillas HTML comunes para el sistema (cabecera, logo, etc.).

La carpeta `pages` corresponde al controlador y a la vista de MVC. Aquí se crean las páginas de la aplicación web y que serán mostrados al usuario según su rol. En cada una de las páginas existirá su correspondiente controlador, además de algunos componentes web que fueron creados para *frontend*. Dichos componentes, son gestionados como recursos estáticos desde la página mediante el archivo descriptor.

La carpeta `handlers` corresponde a una parte del controlador de MVC. Esto es porque no están asociados a ninguna página.

a.1 Carpeta Handlers

Hay varios *handlers* que no pueden ser asociados a una página en particular, pero tienen el mismo comportamiento que un *handler* asociado a una página (en la subsección se describe acerca de un *handler* que corresponde a una página). Esto es, manejar las peticiones que son realizadas por el cliente web ya sea utilizando AJAX o *WebSocket*. Este tipo de peticiones son manejadas en *handlers* dentro de la carpeta con el mismo nombre como se muestra en la Figura 2.42.

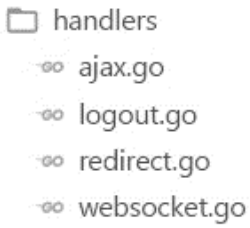


Figura 2.42. Contenido de la carpeta `handlers`

El archivo `logout.go` es utilizado para terminar la sesión del usuario.

El archivo `redirect.go` se utiliza para redireccionar de una página a otra, según lo haya solicitado un handler de cualquiera de las páginas de la aplicación web.

A continuación, se realiza una revisión más detallada tanto del archivo `ajax.go` y `websocket.go` puesto que contienen métodos importantes para manejar peticiones.

En el archivo `ajax.go` se encuentran dos métodos que son utilizados para obtener información directamente desde la base de datos y devolver el resultado en formato JSON. En el Código 2.7, se puede ver el contenido del método `GetStatisticsData`. Este método, permite obtener toda la información estadística que será mostrada en el

panel de administración. Cada uno de los métodos mostrados en dicho código, se comunican directamente con la capa Modelo, la cual se encargará de conectarse a la base de datos y realizar las consultas correspondientes para obtener la información deseada.

El resultado es devuelto al cliente en formato JSON (línea 31) el cual contiene la información que fue solicitada, si es que no ha ocurrido algún error, o si existe dicha información. Esta función se utiliza para presentar las estadísticas de usuarios registrados (línea 18 y 19), problemas registrados (líneas 20 y 21), equipos registrados (líneas 24 y 25), entre otros, en el panel de administración general del sistema.

```
12  // GetStatisticsData ...
13  func GetStatisticsData(res http.ResponseWriter, req *http.Request) {
14
15      var data map[string]int
16      val := bone.GetValue(req, "type")
17      switch val {
18      case "all-users":
19          data = r.GetUsers()
20      case "all-challenging-problems":
21          data = r.GetProblems()
22      case "all-tourneys":
23          data = r.GetStatictistTourneys()
24      case "all-teams":
25          data = r.GetTeams()
26      case "all-universities":
27          data = r.GetUniversities()
28      case "all-carriers":
29          data = r.GetCarriers()
30      }
31      u.ResponseJSON(res, data)
32  }
```

Código 2.7. Método `GetStatisticsData` del archivo `ajax.go`

En el Código 2.8, se muestra el método `GetInfomation` que es utilizado para responder a peticiones desde cualquier rol de usuario. Por ejemplo, para devolver información acerca del actual torneo (líneas 54 a 56), datos de un usuario (líneas 51 a 53) entre otros.

En el Código 2.9 se presenta parte del código contenido en el archivo `websocket.go` que contiene el método `EvaluateCode` y se utiliza para realizar la evaluación del código cuando un usuario envía la solución a un problema planteado.

El código contenido en el bucle `for` (línea 30) es utilizado para permanecer escuchando por el *WebSocket* mientras dura la conexión entre el cliente y el servidor. El código de la línea 34 permite escuchar por la información que es enviada desde el cliente. Dicha

información se asigna a la variable `msg` que se pasa como referencia al método `ReadJSON`. La información de la variable `msg` se asocia a la estructura de datos `info` (línea 41 a 46). Esta estructura contiene el formato que el servicio web requiere para evaluar el código enviado desde la aplicación web. La información contenida en la estructura de datos `info` es convertida en un objeto JSON para ser pasada a la utilidad `make_request.go` (líneas 47 y 49).

```
34  //GetInformation ...
35  func GetInformation(res http.ResponseWriter, req *http.Request) {
36      var data interface{}
37      val := bone.GetValue(req, "type")
38      switch val {
39      case "langs":
40          data, _ = r.GetLangs()
41      case "data-types":
42          data, _ = r.GetDataTypes()
43      case "levels":
44          data, _ = r.GetLevels()
45      case "all-challenging-problems":
46          data, _ = r.GetListProblems()
47      case "all-universities":
48          data, _ = r.GetListUniversities()
49      case "all-carriers":
50          data, _ = r.GetListCarriers()
51      case "user-data":
52          query := bone.GetQuery(req, "username")
53          data, _ = r.GetUsername(query[0])
54      case "current-challenge":
55          teamName := bone.GetQuery(req, "team-name")
56          data, _ = r.GetInfoNowTourneyByUserID(strings.Join(teamName, ""))
57      case "challenge":
58          id := bone.GetQuery(req, "id")
59          data, _ = r.GetProblemsTourneyBy(id[0])
```

Código 2.8. Método `GetInformation` del archivo `ajax.go`

El resultado de la evaluación será almacenado en la variable `result` (línea 48). Además, el servicio web retorna un código de estado que es asociado a la respuesta. Si el código de estado es 200, entonces la evaluación del código se ha ejecutado con éxito y, por lo tanto, la variable `result`, contiene dicho resultado. Este resultado se agrega a la estructura de datos llamada `evaluation` (línea 51). En las líneas 54 a 56, se utiliza el método `SaveTestResult` para almacenar el resultado del problema planteado en la tabla `TestResult`. En la línea 57, utilizando el método `WriteJSON` del *WebSocket* `wsContestant`, el mismo resultado almacenado, ahora es pasado al cliente para que se muestre en la consola de resultados en el área de torneos de programación.

Después de llevar a cabo estas acciones, se debe retornar el control de la aplicación a la Goroutine principal, sin embargo, queda una tarea más por realizar. Esta tarea está asociada con el cálculo del valor final utilizando los resultados parciales que se van registrando en la tabla `TestResult`. A partir de la línea 58 a la línea 60, se trabaja con una Goroutine como un proceso en *background* para obtener el promedio final y que posteriormente sea almacenado dicho resultado en la tabla `Score`.

```
30  func EvaluateCode(w http.ResponseWriter, r *http.Request) {
31      wsContestant, err := upgrader.Upgrade(w, r, nil)
32      for {
33          var msg st.Evaluate
34          err := wsContestant.ReadJSON(&msg)
35          type Data struct {···
40          }
41          info := &Data{···
46          }
47          jsonMessage, _ := json.Marshal(info)
48          codeStatus, result, _ := u.Make.Request("POST").Test()
49                                       .WithData(jsonMessage)
50          if codeStatus == 200 {
51              var evaluation st.Evaluation
52              ...
53              ...
54              err := repo.SaveTestResult(
55                  evaluation, msg.ID,
56                  msg.TeamID, msg.PartialTime)
57              wsContestant.WriteJSON(&evaluation)
58              go func() {
59                  repo.CalculeFinalScore(msg.TeamID, msg.TourneyID)
60              }()
61          }
62
63      }
64  }
```

Código 2.9. Método para devolver información vía *WebSocket*

Este proceso se va realizando cada vez que el usuario envía la solución a un problema planteado. El valor calculado es parcial y sigue siendo almacenado en la tabla `Score`. Cada vez que se registra un nuevo resultado parcial, el último valor registrado en la tabla `Score` se actualiza.

a.2 Carpeta pages

Todas las páginas que han sido creados en el sistema tienen la misma estructura de archivos. En la Figura 2.43 se puede ver la estructura de carpetas existentes en la carpeta `pages`.

La página `admindash` corresponde a la página de administración general del sistema.

La página `carrier` corresponde a la página que mostrará el formulario para registrar carreras universitarias en el sistema.

La página `challenge` corresponde a la página que mostrará un formulario para crear problemas para ser resueltos.

La página `confirmation`, es una página que se usa para mostrar los mensajes de confirmación cuando se ha enviado un correo ya sea porque se registra un usuario o se recupera una contraseña o se ha realizado la invitación a formar parte de un equipo para participar en un torneo.

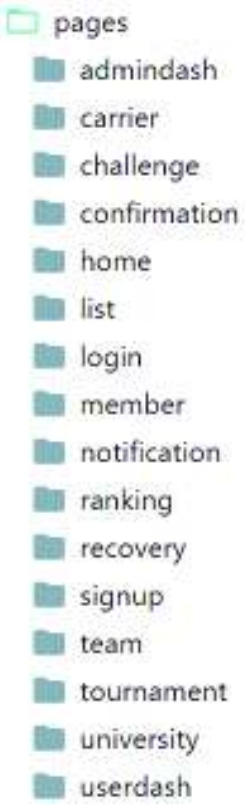


Figura 2.43. Páginas de la aplicación web

La página `home` es la página principal del sistema.

La página `list`, es una página genérica que se utiliza para mostrar la lista de contenidos ya sea de usuarios registrados, ejercicios creados, torneos programados, etc.

La página `login` es usada para mostrar el formulario de inicio de sesión a la aplicación web.

La página `member` se utiliza para mostrar el formulario que debe llenar un usuario que va a formar parte de un equipo.

La página `notification` se utiliza para mostrar los mensajes de error 200, 400, 500, etc.

La página `ranking` corresponde a la página que muestra la tabla de posiciones del último torneo organizado.

La página `signup` se utiliza para mostrar el formulario de registro de nuevos usuarios a la aplicación web.

La página `recovery` muestra un formulario de recuperación de contraseña.

La página `team` muestra un formulario para que un usuario registrado pueda crear un equipo y realice la invitación a otros usuarios a formar parte de su equipo.

La página `tournament` se utiliza para mostrar el espacio para los torneos de programación.

La página `university` muestra un formulario para registrar nuevas universidades al sistema.

La página `userdash` muestra secciones de próximos torneos, notificaciones y enlace para acceder a un torneo, considerando todos los requerimientos que debe cumplir un usuario para poder acceder a dicho torneo.

Cada una de las páginas dentro de la carpeta `pages` tienen la misma estructura, como se ve en la Figura 2.44.

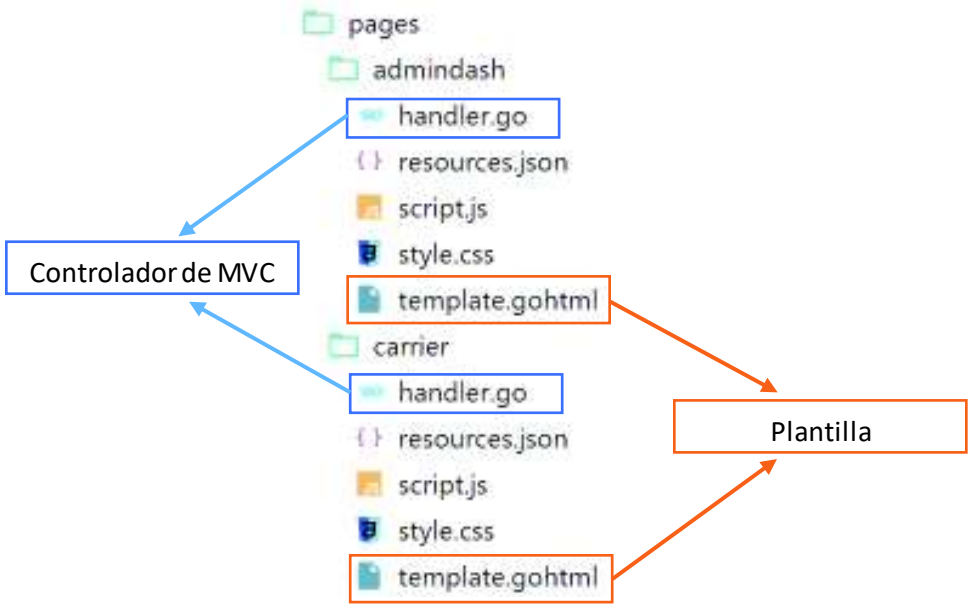


Figura 2.44. Estructura de una página

El archivo `script.js` contiene código Javascript para agregar funcionalidades a la parte visual de la página web en cuestión. Su ámbito viene dado únicamente a dicha página, es decir, cualquier código creado dentro de `script.js`, solo podrá ser utilizado por la página que contiene dicho archivo.

Por otro lado, el archivo `style.css` contiene estilos que son únicamente para la página que contiene dicho archivo. Este archivo no necesariamente contiene estilos, sin embargo, si se agregan estilos a dicho archivo, su ámbito está definido únicamente para la página que contiene dicho archivo.

El archivo descriptor `resources.json` tiene la misma estructura para todas las páginas. En el Código 2.10 se muestra el archivo descriptor que está asociado a la página de `admindash`.

Como se puede observar, todas las dependencias de archivos estáticos internos que se encuentran dentro de la carpeta `static`, y externos que se obtienen a través de una URL provista, son indicadas en este documento.

Estos archivos estáticos serán cargados en el resultado de renderizar el `template` de la página.

```
{
    "css": [
        "/static/css/app.05c05417c52c5e18749024429f450cf0.css"
    ],
    "js": [
        "https://cdnjs.cloudflare.com/ajax/libs/axios/0.18.0/axios.min.js",
        "https://code.jquery.com/jquery-3.2.1.slim.min.js",
        "/static/js/manifest.af6e8fe3a67da10ae9b7.js",
        "/static/js/vendor.6361ba9c252a6893fe07.js",
        "/static/js/app.3e8bcd30ddec04760385.js"
    ],
    "partials": ["header", "simple-footer", "topmenu", "logo"]
}
```

Código 2.10. Archivo descriptor de la página `admindash`

El archivo `handler.go`, corresponde al controlador de cada página y tiene dos métodos con nombres `Get` y `Post` que son bastante descriptivos (Código 2.11); permiten manejar las peticiones de los usuarios ya sea para solicitar contenido (`Get`), guardar o actualizar información (`Post`).

En el método `Get`, generalmente llama al `template` que está contenido en el archivo `template.go` mediante la utilidad `render.go` para generar el contenido que será mostrado al usuario. Un ejemplo del método `Get` se puede ver en el Código 2.12. La

línea 18, corresponde a la inicialización de la estructura de datos, la cual contendrá la información que será pasada al `template` (línea 35) para ser procesada y mostrada al usuario como contenido final que es renderizado en el servidor.

```
11    // Get ...
12  ⊞ func Get(res http.ResponseWriter, req *http.Request) {···
30    }
31    // Post ...
32  ⊞ func Post(res http.ResponseWriter, req *http.Request) {···
52    }
```

Código 2.11. Código genérico para los *handlers* de cada página

En la línea 20, se obtiene una variable asociada a la petición `Get` para posteriormente ser utilizada según corresponda la funcionalidad de la página. Las líneas 21 a 25, corresponden a una funcionalidad que permite redireccionar a la página de notificación, si el usuario quiere acceder a una página sin tener el rol requerido.

La sentencia `switch` se encarga de comparar el valor de la variable que fue pasada en la URL y establecerá información en la estructura de datos mencionada anteriormente. El código de la línea 35, se encargará de llamar al `template` asociado a la página y renderizar el contenido para finalmente ser mostrado al usuario.

```
13  var title = "Panel de administración"
14
15  // Get : Página principal
16  func Get(res http.ResponseWriter, req *http.Request) {
17
18      info := s.DataView{}
19      info.Title = title
20      val := bone.GetValue(req, "type")
21      h.Allow.
22          IfIs("admin").
23          AddDataTo(&info).
24          IfNot(res, req).
25          RedirectTo("")
26
27      switch val {
28      case "dashboard":
29          info.Title = "Panel de administración"
30          info.Render = "dashboard"
31      case "settings":
32          info.Title = "Panel de configuración"
33          info.Render = "settings"
34      }
35      u.Render(res, "admindash", info)
36  }
```

Código 2.12. Método `Get` del `handler` de la página de `admindash`

En el Código 2.13, se muestra un ejemplo del método `Post` asociado al archivo `handler.go` de la página *Carrier*. Generalmente, el método `Post` en cada archivo `handler.go` de cada página de la aplicación web, tendrán líneas de código para capturar información que es enviada desde un formulario web (líneas 39 a 41), para ser almacenada o actualizada utilizando métodos del modelo que están disponibles en los archivos dentro de la carpeta `repositories`.

En la línea 43 se puede observar que el valor de la variable `name` capturada del formulario web, se pasa como argumento al método `SaveCarrier` del paquete `repositories`, que corresponde a la carpeta con el mismo nombre.

El resultado devuelto corresponde a un error que es comparado en la línea 45. Si no existe error, no se agrega ninguna información a la estructura de datos declarada en la línea 38. Caso contrario, si se generó algún error en el instante de almacenar la información en la base de datos, la variable `err`, contendrá información del error. Esta información será asociada a la estructura de datos `errors` para posteriormente ser pasada a la vista y mostrado al usuario.

El mismo mecanismo de renderizado utilizado en el método `Get`, se utiliza en el método `Post` (línea 56).

```
36  // Post ...
37  func Post(res http.ResponseWriter, req *http.Request) {
38      errors := make([]string, 0)
39      req.ParseForm()
40
41      name := req.Form.Get("name")
42
43      err := r.SaveCarrier(name)
44
45      if err != nil {···
47      } else {···
54      }
55      info := &s.DataView{Title: title, Errors: errors}
56      u.Render(res, "carrier", &info)
57  }
```

Código 2.13. Método `Post` del *handler* de la página `carrier`

a.3 Carpeta repositories

Dentro de la carpeta `repositories` se encuentran todas aquellas utilidades que se asocian con una tabla de la base de datos. Los métodos del modelo se almacenan en esta carpeta.

Cada archivo tiene métodos que realizan una consulta a la base de datos. Dichas consultas, están basadas en el estándar SQL (*Structured Query Language*) sin ninguna utilidad que mapee o relacione los métodos con las tablas. Por lo tanto, solo se requieren las librerías estándar de Go para conectarse a la base de datos MySQL y realizar las consultas SQL. Estas librerías son: `"database/sql"` y `"github.com/go-sql-driver/mysql"` y son importados mediante la utilidad `mysql.go` ubicado en la carpeta `utils`.

Dado que la mayoría de consultas son, por lo general, para solicitar información o guardar o actualizar registros, no se realiza ningún análisis en particular. Sin embargo, dado que el cálculo del resultado final para equipo se lo realiza mediante una consulta, a continuación, se realiza un análisis de los métodos que son utilizados para llevar a cabo dicha acción.

A continuación se presenta parte del código del archivo `score.go` que es utilizado para calcular el resultado final de evaluación par acada equipo. El Código 2.14 se encuentra en el archivo `score.go` dentro de la carpeta `repositories`. Hay dos métodos que son utilizados tanto para registrar el cálculo parcial de la información, así como el cálculo total de los resultados enviados por el servicio web. Esta utilidad es llamada desde `websocket.go`.

```
3   func CalculeFinalScore(teamID string, tourneyID int) error {
4
5       query := `SELECT
6       ROUND(100*SUM(tr.passed)/SUM(tr.number_tests), 3) AS average,
7       COUNT(tr.number_tests) AS totalProblems,
8       ROUND((ROUND(100*SUM(tr.passed)/SUM(tr.number_tests), 3)
9       *COUNT(tr.number_tests)/100), 0) AS solvedProblems,
10      ROUND(SUM(tr.time_compilation), 4) AS totalTimeCompilation,
11      MAX(tr.time_interval) AS timeInterval
12      FROM TestResult AS tr
13      INNER JOIN TournamentChallengingProblem AS tcp ON
14      tcp.id=tr.TournamentChallengingProblem_id
15      WHERE tcp.Tournament_id=` + strconv.Itoa(tourneyID) +
16      `  AND tr.Team_id=` + teamID + `;`
17
18      db := u.OpenDB()
19      result, err := db.Query(query)
20
21  }
22  func updateScore(s *s.Score) error {···
45  }
```

Código 2.14. `score.go` para calcular, almacenar y actualizar el *ranking*

La función `CalculeFinalScore` se encarga de realizar el cálculo basado en las calificaciones parciales de los ejercicios resueltos por un equipo, en un torneo que se ha llevado a cabo.

Los resultados parciales se encuentran en la tabla `TestResult`. Y el resultado final será almacenado en la tabla `Score` utilizando la función privada `updateScore` (las líneas 5 a 16).

Los datos que se necesitan para ser almacenados en la tabla `Score` corresponden a:

`FinalScore`: Corresponde al valor final obtenido

`TotalProblems`: Corresponde al número de problemas planteados

`SolvedProblems`: Corresponde al número de problemas resueltos

`TotalTimeCompilation`: Corresponde a la suma parcial de tiempos de compilación para cada ejercicio resuelto.

`FinalTimeTournament`: Corresponde al tiempo total que ha transcurrido desde el inicio del torneo hasta que finalizó la resolución de los ejercicios propuestos. Estos datos son tomados de la tabla `TestResult` que contiene los resultados de cada ejercicio.

El valor de `FinalScore` se calcula usando la formular presentada en la Ecuación 2.1.

$$Round\left(\frac{\sum pruebas\ parciales}{\sum número\ de\ pruebas} * 100\right)$$

Ecuación 2.1. Cálculo del valor para `FinalScore` de la tabla `Score`

El valor para `TotalProblems` se calcula únicamente contando los problemas basado en el identificador asociado.

El valor para `SolvedProblems` es calculado mediante la Ecuación 2.2.

$$Round\left(Round\left(\frac{\sum pruebas\ parciales}{\sum número\ de\ pruebas}\right) * número\ de\ pruebas\right)$$

Ecuación 2.2. Cálculo del valor para `TotalProblems` de la tabla `Score`

Para obtener el valor del tiempo de compilación (`TotalTimeCompilation`) y tiempo utilizado en la resolución de problemas (`FinalTimeTournament`), simplemente se suma los tiempos parciales registrados en la tabla `TestResult` para cada problema.

b. Componentes web

En esta sección se describen algunos de los componentes web que han sido desarrollados para ser agregados a las plantillas de las páginas de la aplicación si asi lo requieren. Es importante mencionar que estos componentes son desarrollados de forma independiente utilizando la herramienta Vue-webpack-boilerplate.

Después de terminado el desarrollo de cada componente, y, haciendo uso de la misma herramienta, se generan recursos estáticos finales como empaquetados Javascript y CSS para que puedan ser agregados en la plantilla que corresponda dentro de cada página.

Estos recursos generados como empaquetados son: una hoja de estilo del diseño del componente web, un *script* con extensión `.js` que contiene el componente web y toda su funcionalidad y, un *script* con extensión `.vendor.js` que contiene las dependencias del componente web.

En el Código 2.15, se muestra un ejemplo de cómo se debe agregar los empaquetados generados por la herramienta Vue-webpack-boilerplate, a un *template* de una página.

```
1  <html>
2      <head></head>
3      <style href="css_componente_web_tournament_workspace"></style>
4      <body>
5        <tournament-workspace></tournament-workspace>
6        <script src="script_componente_web_tournament_workspace.js" ></script>
7        <script src="script_componente_web_tournament_workspace.vendor.js"></script>
8      </body>
9  </html>
```

Código 2.15. Componente web `tournament-workspace` en una página HTML

Como se observa en la línea 3, se indica la ruta de la hoja de estilo del componente web. En la línea 5, se especifica el componente web como una etiqueta HTML. En la línea 6 se indica la ruta del código fuente del componente web y en la línea 7, se indica la ruta de las dependencias de dicho componente.

Este proceso se realiza para todos los componentes que han sido desarrollados para el este prototipo. Es importante indicar que se han implementado 4 componentes web. Estos componentes son: `repository-workspace`, `event-workspace`, `user-form` y `tournamente-workspace`.

A continuación, se analiza de forma general el componente web `tournament-workspace`. Posterior a este análisis, se realizará una breve descripción de los `event-workspae` y `repository-workspace`.

El componente web `tournament-workspace` tiene diferentes funcionalidades y características integradas, las cuales se listan a continuación:

- Navegación entre problemas propuestos para el torneo que se esté llevando a cabo.
- Editor de código.
- Espacio para mostrar el resultado de las pruebas ejecutadas para cada problema.
- Soporte para comunicación vía AJAX para solicitar información que será cargada al componente.
- Soporte para comunicación vía *WebSocket* para comunicarse con el servidor de la aplicación web.
- Soporte de un temporizador que indica el tiempo que resta para terminar la competición.

Para proveer todas las funcionalidades anteriormente mencionadas, es necesario realizar una modularización que permita separar las responsabilidades. La modularización realizada en el componente web `tournament-workspace` se muestra en la Figura 2.45.

Figura 2.45. Estructura del componente web `tournament-workspace`

Los componentes hijos de `tournament-workspace` son: `countdown`, `spinner` y `codemirror` los cuales están almacenados en los archivos `CountDown.vue`, `Spinner.vue` y `Codemirror.vue`, respectivamente (Figura 2.45).

Es importante mencionar que Javascript cuenta con un objeto global llamado `window` que contiene todos los objetos base Javascript y al que se pueden agregar nuevos objetos. El archivo `main.js` contiene código para cargar las librerías que son requeridas, también contiene configuraciones que serán pasadas al componente principal llamado

`tournament-workspace`, además, contiene código que permite realizar una petición al servidor solicitando los datos del torneo que actualmente se está ejecutando.

Las configuraciones mencionadas están asociadas con detalles de:

- Las URL de petición para AJAX y *WebSocket*.
- Los parámetros de configuración para el editor de código Codemirror.
- Los datos de usuario, datos del torneo, datos del equipo y lenguaje de programación.

Las configuraciones mencionadas anteriormente, deben ser pasadas creando un objeto llamado `tws`, dentro del objeto `window` de Javascript como se muestra en el Código 2.16. En la línea 3, se crea el objecto `tws` dentro del objeto global `window` de Javascript. Este nuevo objeto, tendrá varios objetos anidados.

A continuación, se describe de forma general el uso de cada parámetro de configuración del objeto `tws`.

- `evaluate[string]`: corresponde a la URL a la cual apuntará el componente para enviar el código a ser evaluado (línea 6).
- `data[string]`: corresponde a la URL a la cual apuntará el componente para solicitar los datos del torneo que se está ejecutando (línea 7).
- `redirectByFinalizedTourney[string]`: corresponde a la URL que permitirá que se realice la redirección a una nueva página cuando haya finalizado el torneo, independientemente si se resolvió o no la solución de todos los ejercicios propuestos (línea 8).
- `redirectBySolvedProblems[string]`: igual que la anterior, permitirá redireccionar cuando los ejercicios se hayan completado (línea 9).
- `type[string]`: corresponde al tipo de método HTTP que se utilizará para comunicarse con el servidor. Por ejemplo, POST, GET, etc. En este caso será el método POST (línea 11).
- `websocket[object]`: Es un objeto Javascript que contiene parámetros de conexión como el protocolo, la URL y el puerto para la conexión con *WebSocket* (línea 12 a 15).

- `dataUser[object]`: Contiene información acerca de los datos del usuario, del equipo, identificador del torneo actual y el lenguaje de programación asociado al equipo (línea 18 a 23).

```
1  <tournament-workspace></tournament-workspace>
2  <script>
3      window.tws = {}
4      tws.conn = {
5          url: {
6              evaluate: '/evaluate-code',
7              data: '/get-information/current-challenge',
8              redirectByFinalizedTourney: '/',
9              redirectBySolvedProblems: '/'
10         },
11         type: 'POST',
12         websocket: {
13             protocol: 'ws',
14             url: 'localhost',
15             port: '3000'
16         }
17     }
18     tws.dataUser = {
19         teamID: 2,
20         username: 'username',
21         language: 'CSharp',
22         tourneyID: 1
23     }
24 </script>
```

Código 2.16. Datos de configuración para `tournament-workspace`

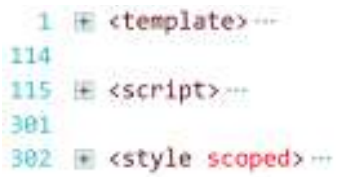
Los datos de configuración contenidos en el objeto `tws` son pasados como propiedades en el objeto Vue como se ve en la línea 55 del Código 2.17.

```
51 new Vue({
52     el: 'tournament-workspace',
53     render(h) {
54         return h(App, {
55             tws: window.tws
56         })
57     }
58 })
```

Código 2.17. Archivo `main.js` del componente `tournament-workspace`

En el archivo `App.vue`, como se muestra en el Código 2.18, se encuentra código HTML dentro de la etiqueta de apertura y cierre `template` (líneas 1 a 113), código Javascript dentro de la etiqueta `script` (líneas 115 a 300) y etiquetas de estilo CSS dentro de la

etiqueta `style` (líneas 302 en adelante). El atributo `scoped`, antes de la llave de cierre de la etiqueta `style` (líneas 302), permite que todos los estilos sean aplicados solo al componente en cuestión.



Código 2.18. Archivo `App.vue` del componente `tournament-workspace`

El método `catchTestSolution` (Código 2.19), que está contenido dentro de la etiqueta `script` (Código 2.18), permite capturar la información que ha sido ingresada en el editor de texto.

La línea 201 del Código 2.19, tiene como propósito, verificar si el botón, para enviar la solución de un problema, está desactivado; si lo está, no se puede enviar otro problema hasta que no se haya recibido la evaluación del problema ya enviado.

Cuando no está desactivado, el primer paso es desactivar el botón (línea 202) para que el competidor no pueda enviar otro problema, hasta que el actual haya sido evaluado en el lado del servidor.

La línea 203, verifica si aún existen problemas que aún no han sido enviados para ser evaluados. En caso de que existan, se captura la información del actual código contenido en el editor de código (línea 206).

La línea 207, con el método `splice`, se elimina el problema del *array* que contiene todos los problemas propuestos que aún no son solucionados.

El objeto `data` (líneas 208 a 217) contiene la información necesaria, que será enviada al servidor. Esta información contenida en `data` corresponde a

`Id`: identificador del problema (línea 209)

`code`: estructura del código resultante que contiene el algoritmo escrito por el equipo participante (línea 210)

`group`: el nombre del equipo participante (línea 214)

`method`: nombre del método del problema (línea 215)

`language`: el lenguaje que el equipo escogió para solucionar los problemas

En la línea 218, se llama al método `sendViaWebSocket` que recibe como parámetros el objeto `data` y un método *callback* que será llamado cuando se reciba el resultado desde la aplicación web. Cuando la aplicación web devuelva la respuesta, se llama al *callback* y empieza la ejecución del código de las líneas 219 a 231.

```
199   catchTestSolution: function () {
200     this.showModal = false
201     if (this.sendEnableButton !== 'button disable-sendbutton') {
202       this.sendEnableButton = 'button disable-sendbutton'
203       if (this.data.challenges.length > 0) {
204         this.lastChallengeTitle = this.currentChallengeTitle
205
206         this.data.challenges.splice(this.currentChallengeNumber - 1, 1)
207         this.allChallengesValue.splice(this.currentChallengeNumber - 1, 1)
208         const data = {
209           id: this.data.id,
210           code: `public class ` + this.dataUser.language + `
211           Temp {\n\t public static ` + this.currentCode.replace(
212               this.currentNameMethod, `evaluatesExpression`) +
213           `\n}`,
214           group: this.dataUser.groupName,
215           method: this.currentMethod,
216           language: this.dataUser.language
217         }
218         this.sendViaWebSocket(data, (data, err) => {
219           if (!err) {
220             this.spinnerConsole = false
221             this.sendEnableButton = 'button light-button'
222           }
223           if (data !== null) {
224             this.results.push(data.response)
225             //this.setSessionStorage()
226             setTimeout(() => {
227               if (this.gameover > 0) {
228                 this.gameover--
229               }
230             }, 100)
231           }
232         })
233       }
234     }
235   },
```

Código 2.19. Método para capturar el algoritmo en el editor de código

En la línea 224, la respuesta recibida se agrega al *array* `results` para que posteriormente sea procesado para ser presentado en la consola de resultados. El objeto `setTimeout` de la línea 226, permite reducir una variable que es un indicador para que se active el mecanismo de redirección. El Código 2.20, muestra parte del código del componente `countDown` que se encuentra en el archivo `CountDown.vue` dentro de la carpeta `components` (Figura 2.45). Este código se utiliza cuando ha finalizado el

temporizador del torneo. La línea 44, activa un evento con el objeto `$emit` que está contenido dentro del objeto `self`, para avisar al componente padre `tournament-workspace`, que el tiempo ha finalizado.

```
42          if (timeInterval < 0) {
43            self.$emit('deadline', 'gameover')
44            clearInterval(identifier)
45          }
```

Código 2.20. Porción del código del componente web `countDown`

En la Figura 2.46, se muestra un diagrama de actividades y las acciones que va tomando el componente `tournament-workspace`, según el resultado obtenido en cada acción.

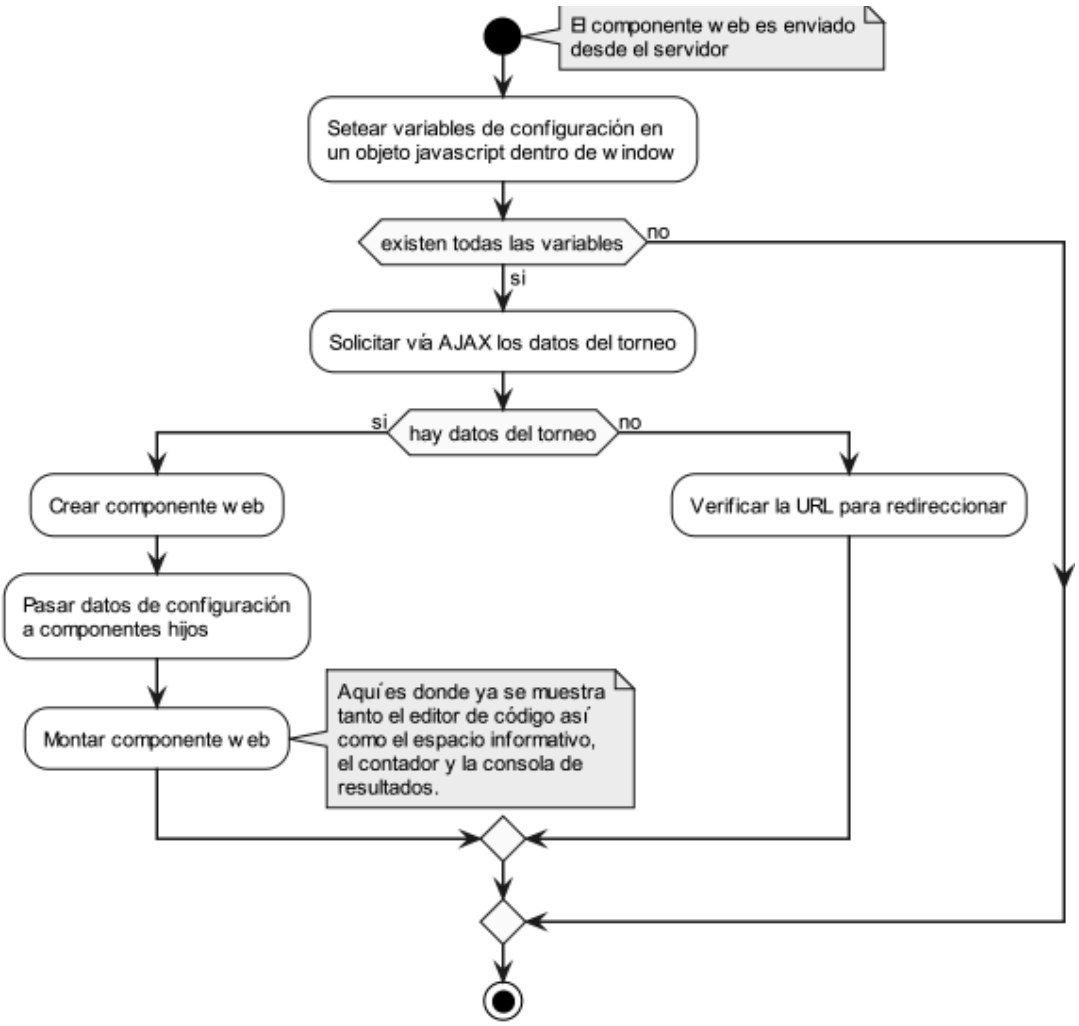


Figura 2.46. Diagrama de actividades componente `tournament-workspace`

El primer parámetro que recibe el objeto `$emit` es una función llamada `deadline`, y cómo segundo parámetro, el *string* `gameover`, que se utiliza como un identificador de que el tiempo terminó. El objeto `clearInterval` detiene el temporizador del torneo.

En la Figura 2.47, y en la Figura 2.48, se muestran diagramas de secuencia de cómo es el proceso de inicialización del componente `tournament-workspace`, y como es la interacción entre dicho componente y la aplicación web.

Figura 2.47. Ciclo de inicialización del componente `tournament-worspace`

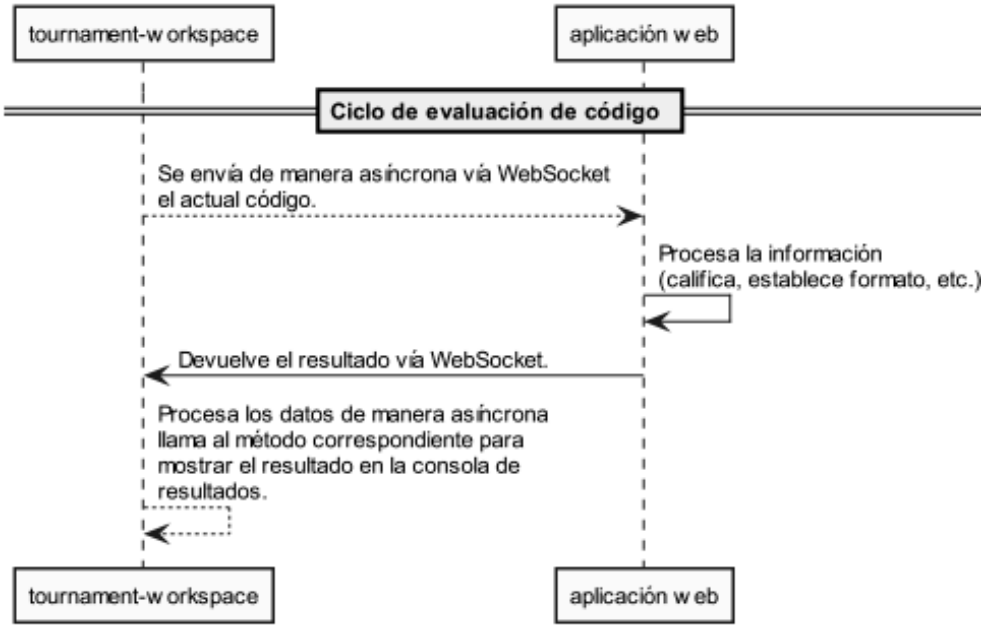


Figura 2.48. Interacción entre `tournament-workspaces` y aplicación web

La interfaz del componte web `tournament-worspace` se puede ver en la Figura 2.49.

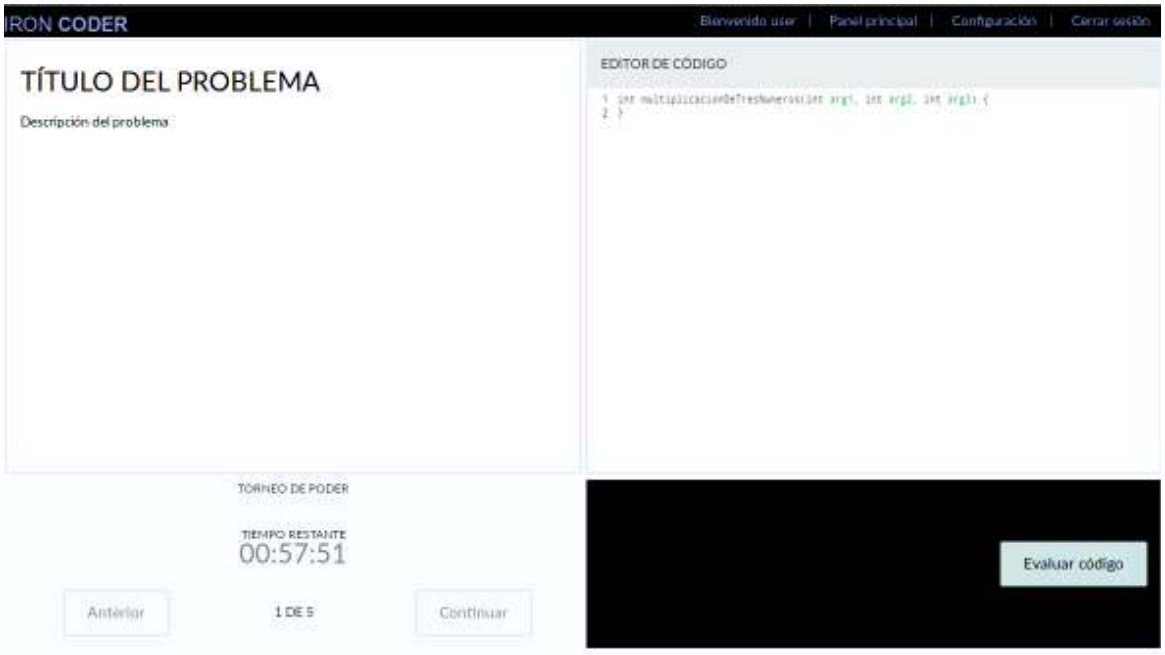


Figura 2.49. Interfaz del componente web `tournament-worspace`

A continuación, se detallan algunos aspectos importantes que han sido desarrollados para el componente web `repository-workspace`. Para ello, se muestra la Figura 2.50, que es el resultado final de la interfaz visual del componente en cuestión.

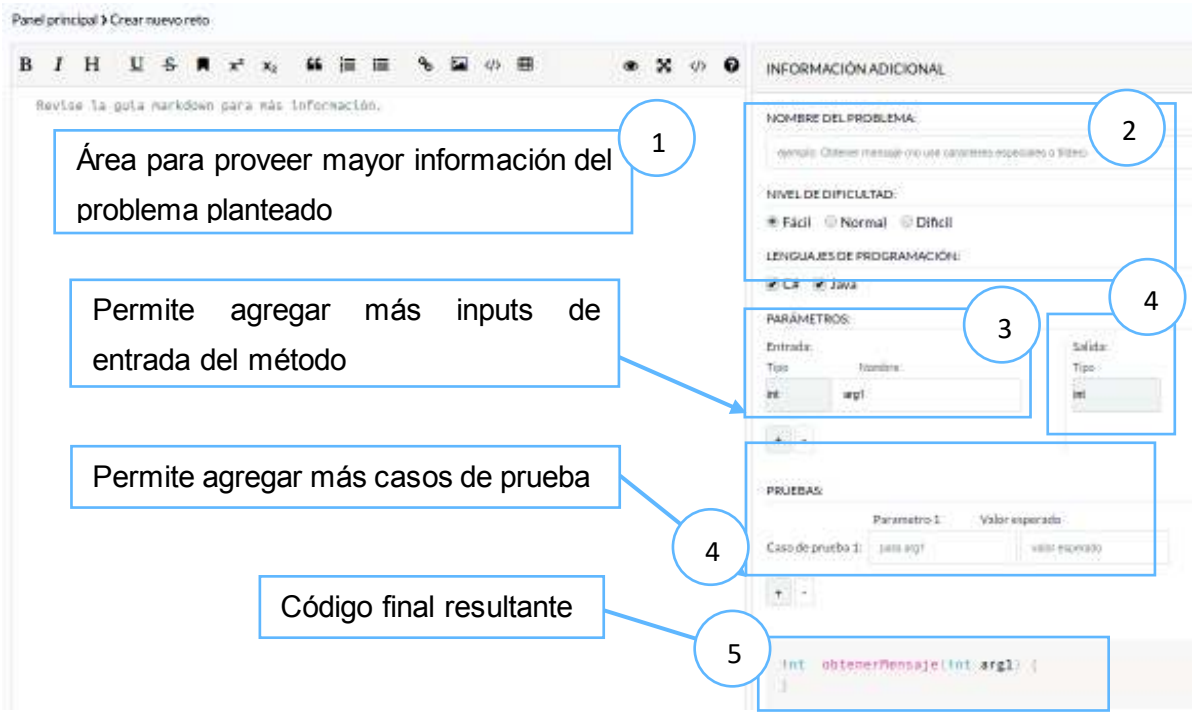


Figura 2.50. Resultado interfaz gráfica de `repository-workspace`

Considerando la Figura 2.50, las funcionalidades implementadas para el componente web `repository-workspace` son:

- Espacio para escribir la información del problema propuesto. Se utiliza un editor de texto Markdown (1).
- Campos estáticos para llenar información acerca de (1):
    - Nombre del método
    - Lenguaje de programación
    - Nivel de dificultad del problema
- Campos dinámicos para llenar información acerca de:
    - Hasta 3 parámetros de entrada para definir tipo y valor de los argumentos del método a ser evaluado (3)
    - Parámetro de salida para definir tipo y valor que se espera (4)
    - Campos para llenar hasta 3 pruebas unitarias teniendo en cuenta la cantidad de parámetros de entrada (4)
- Validación de cada uno de los campos.
- Espacio para mostrar el código resultante que será pasado en las competiciones (5).

La estructura de archivos de este componente se muestra en la Figura 2.51.

Figura 2.51. Estructura de archivos de `repository-workspace`

El archivo `main.js` contiene código que permite importar las dependencias. Estas son: el *plugin* `vee-validate` para validar parámetros en los *inputs* HTML del formulario, el *plugin* `mavon-editor` para disponer de un editor de texto de formato Markdown, el *plugin* `vue-js-modal` para ventanas modales. El *plugin* `vue-resource` para realizar peticiones AJAX. Como se ve en el Código 2.21, los parámetros de configuración que

deben ser pasados al componente web están asociados a las URL y son usadas para obtener información de los lenguajes de programación (línea 18), los niveles de dificultad de los problemas (línea 19), los tipos de datos (línea 20) y para guardar los datos registrados (línea 17).

```
10  <repository-workspace></repository-workspace>
11  <script>
12      // repository-workspace (rws)
13      let common = 'get-information/'
14      window.rws = {}
15      rws.conn = {}
16      rws.conn.host = 'http://localhost:8081'
17      rws.conn.urlSave = 'challenge/save'
18      rws.conn.urlGetLangs = common + 'langs'
19      rws.conn.urlGetLevels = common + 'levels'
20      rws.conn.urlGetDataTypes = common + 'data-types'
21  </script>
```

Código 2.21. Configuración pasada al componente `repository-workspace`

En el archivo `App.vue`, el objeto `watch` (línea 588 del Código 2.22) contiene métodos que observan las acciones de cada uno de los campos HTML que existen en el componente web `repository-workspace` (Figura 2.50). Dicho objeto está dentro del objeto `optionTest` y este, es exportado para que sea utilizado por el objeto `Vue` (línea 629).

Los métodos `inputNameMethod` (líneas 589 a 592) y `outputParamType` (líneas 593 a 595), permiten generar el resultado del código que se mostrará en la sección de código resaltado ((5) de la Figura 2.50).

El método `languages` (líneas 596 a 611) es utilizado para escuchar las acciones de los `checkboxs` HTML y seleccionar el lenguaje de programación.

La línea 597, verificará si es que aún no ha sido seleccionado un lenguaje de programación, en cuyo caso, se procede a agregar información al *array* `errors` del objeto `$validator`, que está asociado a la librería vee-validate.

Dicha información, está asociada con el nombre del campo HTML, el mensaje de error que se desea mostrar al usuario, las reglas que son requeridas, en este caso particular, es un campo obligatorio por lo que se utiliza el *string* `required`.

Las líneas de código 605 a 609, permiten que el error que fue agregado, sea quitado del *array* `errors` y por lo tanto ya no se muestre mensaje de error cuando se haya

seleccionado al menos un lenguaje de programación. El método `description` (líneas 612 a 627), sigue el mismo flujo de verificación que el método `languages`.

En la línea 613, se verifica que la variable `value` esté vacía, esto quiere decir, que aún no se ha ingresado información en el área para proveer mayor información del torneo ((1) de la Figura 2.50).

```
588  optionTest.watch = {
589    'inputNameMethod': function (value) {
590      this.nameMethod = this.camelize(value)
591      this.generateCodeChallenge()
592    },
593    'outputParamtype': function () {
594      this.generateCodeChallenge()
595    },
596    'languages': function (value) {
597      if (this.languages.length === 0) {
598        this.$validator.errors.items.push({
599          field: 'lenguaje',
600          msg: 'Debe seleccionar al menos un lenguaje de programación',
601          rule: 'required',
602          scoped: ''
603        })
604      } else {
605        this.$validator.errors.items.forEach((error, index) => {
606          if (error.field === 'lenguaje') {
607            this.$validator.errors.items.splice(index, 1)
608          }
609        })
610      }
611    },
612    'description': function (value) {
613      if (value === '') {
614        this.$validator.errors.items.push({
615          field: 'descripcion',
616          msg: 'Debe agregar la descripción del problema',
617          rule: 'required',
618          scoped: ''
619        })
620      } else {
621        this.$validator.errors.items.forEach((error, index) => {
622          if (error.field === 'descripcion') {
623            this.$validator.errors.items.splice(index, 1)
624          }
625        })
626      }
627    }
628  }
629  export default optionTest
```

Código 2.22. Código del componente web `OptionTest`

A continuación, se detalla de manera general el componente `event-workspace` que es utilizado para gestionar la creación de torneos.

La estructura de archivos de dicho componente, se muestra en la Figura 2.52.

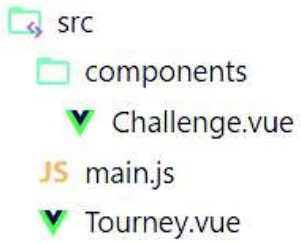


Figura 2.52. Estructura de archivos para el módulo de gestión de torneos

El archivo `main.js`, al igual que los casos anteriores, contiene código que permite pasar los parámetros de configuración al objeto Vue.

El archivo `Tourney.vue` permite generar la interfaz visual mostrada en la Figura 2.53.

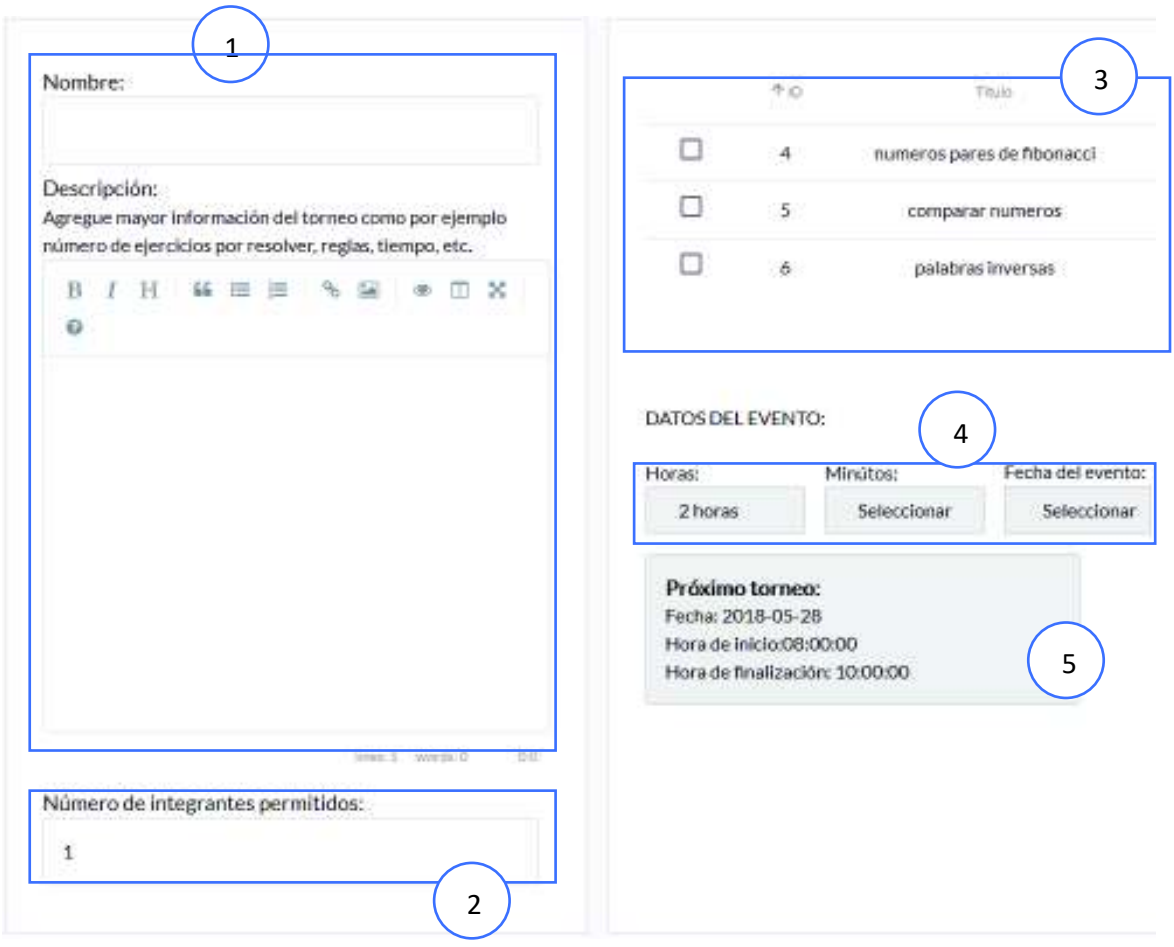


Figura 2.53. Resultado final del espacio para gestionar torneos

Como se muestra en la Figura 2.53, en la parte superior izquierda (1), se encuentra un formulario que permite definir información para el torneo que se desea crear. En la parte inferior izquierda (2), se puede definir el límite de participantes para el torneo que se va a

crear. En la parte superior derecha (3), se muestra una tabla que contiene todos los ejercicios propuestos y que serán asociados al torneo.

El código contenido en `Challenge.vue`. permite generar la tabla con la información de los problemas. En la parte inferior derecha (4), se presenta una serie de elementos HTML `selects` y `buttons` que son usados para definir la fecha y hora del torneo.

Además, se muestra una sección de color gris (5), la cual mostrará la fecha, hora de inicio y hora de finalización que ha sido escogida. Esta sección es netamente informativa y no se puede manipular manualmente la información que contiene. Dicha información cambiará cada vez que cambie la información en los botones para seleccionar horas, minutos, hora de inicio o fecha del evento.

Cuando se da clic en el botón bajo la etiqueta "`Fecha del evento`", se desplegará un cuadro de diálogo que contiene un calendario, como se muestra en la Figura 2.54. Cuando se presiona el botón bajo la etiqueta "`Hora de inicio2`, se desplegará un cuadro de diálogo que contiene un reloj, como se muestra en la Figura 2.55.

Figura 2.54. Modal para fechas

Figura 2.55. Modal para hora

c. Renderizado de templates de cada Vista

En el Código 2.23, de la utilidad `render.go`, se puede observar que existe un método público llamado `Render` (línea 1). Este método, conjuntamente con la librería `html/template`, serán utilizados por cada controlador que necesite renderizar un `template` y que esté asociado a la página (Figura 2.44).

Dicho método recibe 3 argumentos; el primero corresponde a la respuesta HTTP de la petición realizada por el cliente; el segundo, corresponde al nombre de la página dentro

de la carpeta `pages`; el tercero, corresponde a los datos que se quieren renderizar en el `template` de la página, para obtener una vista final que será enviada al usuario.

Además, con ayuda de los *helpers* `css` *y* `js` (líneas 7 a 12), el contenido HTML que fue generado por la utilidad `loadconfig`, será agregado en el renderizado final del `template`.

También se puede observar dos *helpers* adicionales: `string2html` y `add` (líneas 13 a 16). El primer *helper* ayuda a que cualquier contenido HTML que no corresponda a recursos estáticos como `css` o `js`, sea insertado en el `template` de tal manera que sea reconocido como contenido HTML y no como un texto cualquiera cuando es enviado al navegador. Los *helpers* propios del motor de renderizado de Go, conjuntamente con los que han sido creados anteriormente, se encuentran dentro de los archivos `template.gohtml` de cada página.

```
1   func Render(res http.ResponseWriter, namePage string, data
2
3       resources := Generate(namePage)
4       tmpl := resources.Templates
5
6       funcMap := template.FuncMap{
7           "css": func() template.HTML {
8               return template.HTML(resources.Assets.CSS)
9           },
10          "js": func() template.HTML {
11              return template.HTML(resources.Assets.Js)
12          },
13          "string2html": func(content string) template.HTML {
14              return template.HTML(content)
15          },
16          "add": func(value int) int {
17              return value + 1
18          },
19      }
20      t := template.New(namePage)
21      t.Funcs(template.FuncMap(funcMap))
22      t.ParseFiles(tmpl...)
23      t.ExecuteTemplate(res, ".", data)
24  }
```

Código 2.23. Método para renderizar las páginas

En el Código 2.24 se puede ver los *helpers* que son utilizados para cargar todas las dependencias requeridas para renderizar el `template` de una página. El *helper* `define` indica que el `template` actual, está en la misma ruta que el controlador asociado. El *helper* `template` permite que se carguen `templates` parciales que están contenidos

en la carpeta `partials` y que han sido especificados en el archivo descriptor. Los helpers `css` y `js` como se mencionó anteriormente cargarán aquellos recursos estáticos que se encuentran definidos en el archivo descriptor y aquellos que forman parte de la página y que han sido cargados mediante la utilidad `loadconfig.go`

```
1  {{define "."}}
2      {{template "header" .}}
3      {{template "topmenu" .}}
4          //Aquí va todo el código HTML.
5      {{js}}
6      {{css}}
7      {{template "footer"}}
8  {{end}}
```

Código 2.24. Estructura básica de un *template* con *helpers* para cargar recursos

El mecanismo de renderizado final de las páginas es realizado en tiempo de ejecución, pues, si se realizara en tiempo de compilación, el resultado sería contenido estático y, por lo tanto, no sería posible generar contenido que cambie por cada petición y cada usuario que solicite una página.

c.1 Carpeta structs

En la Figura 2.56 se muestra el contenido de la carpeta `structs` que contiene estructuras de datos que han sido definidas tanto para los modelos, así como para la estructura de datos que se da entre una petición y respuesta enviada entre el cliente y la aplicación web y, entre la aplicación web y el servicio web.

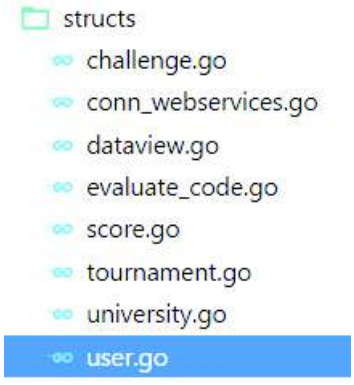


Figura 2.56 Contenido de la carpeta `structs`

En el Código 2.25 se muestra un ejemplo de la estructura de datos definida para el modelo `score`. Como se puede observar, en la parte derecha se define cuales serán los nombres de las *key* de la respuesta JSON que es enviado por el servicio web. Es importante mencionar que, para acceder a cada valor de la estructura de datos, se debe

especificar el nombre siguiento la definición de declaración de métodos y variables según se especifica en Golang, esto es, para declarar públicos los métodos y las variables y, por lo tanto, se pueda acceder desde otras clases, estos deben tener siempre la primera letra del nombre en mayúscula.

```
1   package structs
2
3   //Scores ...
4   type Scores struct {
5       Score []Score
6   }
7
8   //Score ...
9   type Score struct {
10      TourneyName          string  `json:"tourneyName"`
11      TeamID               string  `json:"teamID"`
12      Language             string  `json:"language"`
13      TeamName             string  `json:"teamName"`
14      FinalScore           float32 `json:"finalScore"`
15      TotalProblems        int     `json:"totalProblems"`
16      SolvedProblems       int     `json:"solvedProblems"`
17      TotalTimeCompilation float32 `json:"totalTimeCompilation"`
18      FinalTimeTournament  float32 `json:"finalTimeTournament"`
19      DateTourney          string  `json:"dateTourney"`
20      TournamentID         int     `json:"tournamentID"`
21  }
```

Código 2.25. Estructura para el modelo Score

2.3.3. Servicio web

En esta sección, se describe brevemente la estructura de archivos del servicio web, además, se detalla el mecanismo que se utiliza para ejecutar las pruebas unitarias sobre los algoritmos que son enviados para ser evaluados, utilizando *scripts* de Linux y las librerías JUnit y NUnit.

a. Estructura de archivos

La estructura de archivos y modularización sigue el mismo principio utilizado para crear la aplicación web, con la única diferencia de que en el servicio web no existen modelos ni páginas pues, no una interacción con la base de datos ni tampoco hay una interfaz visual que mostrar.

En la Figura 2.57, se muestra la estructura de archivos del servicio web. En la carpeta `handlers`, se definen una serie de controladores del sistema que son llamados a través de una ruta en particular que ha sido especificada en el archivo `routes.go`. Estos controladores permiten ejecutar tareas para procesar las peticiones que son realizadas

desde el cliente. Esto lo hace con ayuda de las utilidades contenidas dentro de la carpeta utils.

En la carpeta `structs`, están los objetos de Go que contienen estructuras de datos.

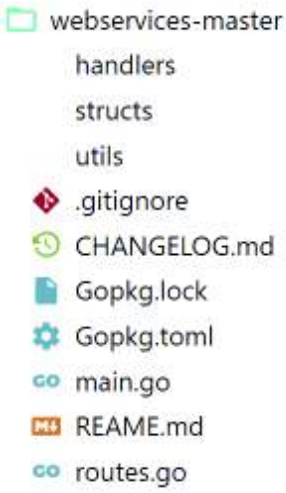


Figura 2.57. Estructura de archivos del servicio web

La carpeta `utils`, como se muestra en la Figura 2.58, contiene varias utilidades que permiten llevar a cabo varias tareas que están enfocadas en el proceso de evaluación de código, creación de espacios de trabajo, creación de archivos con contenidos, entre otros.

Figura 2.58. Estructura de la carpeta `utils` del servicio web

El archivo `auth_validate.go` cuenta con una función llamada `HeaderAuth` que se encarga de validar las peticiones que son realizadas desde la aplicación web (líneas 13 a 30 del Código 2.26). Para ello, verifica un *token* de autenticación que previamente ha sido creado y establecido tanto en el servicio web como en la aplicación web. En el archivo `execute.go`, hay varios métodos, sin embargo, el método `start` es el más importante

pues se encargará de llamar a los *scripts* bash creados para las pruebas unitarias (líneas 46 a 50 del Código 2.27). Además, establecerá un tiempo límite por el cual espera la respuesta del *script*. Este tiempo límite es de 3 segundos (líneas 60 a 68) y es establecido para que la aplicación no quede esperando por una respuesta por un largo periodo de tiempo.

```
12    //HeaderAuth ...
13    func (v validate) HeaderAuth(req *http.Request) error {
14
15        var result string
16        var err error
17
18        if len(req.Header["Ironcoder-Token"]) > 0 {
19            result = req.Header["Ironcoder-Token"][0]
20            if result == ironCoderToken {
21                err = nil
22            } else {
23                err = errors.New("Invalid Token")
24            }
25        } else {
26            err = errors.New("ironcoder-token doesn't exists")
27        }
28
29        return err
30    }
```

Código 2.26. Código de autenticación de peticiones en el servicio web

```
42    func (d data) Start() (string, error) {
43
44        var failed error
45        done := make(chan error, 1)
46        cmd := exec.Command("/bin/sh", "/opt/ironcoder/scripts/execute_"+
47            d.Language, d.NameGroup, d.NameProblem)
48        randomBytes := &bytes.Buffer{}
49        cmd.Stdout = randomBytes
50        err := cmd.Start()
51
52        if err != nil {
53            log.Fatal(err)
54        }
55        go func() {
56            done <- cmd.Wait()
57        }()
58
59        select {
60        case <-time.After(3 * time.Second):···
68        case err := <-done:···
74        }
75    }
```

Código 2.27. Código para crear llamar a los *scripts* de pruebas

Parte del Código 2.28, está contenido en el archivo `generate_code.go` y permite crear el *script* que será utilizado para las pruebas unitarias.

```
26  func createTestJava() string {···
69  import org.junit.Test;
70  public class JavaTempTest {
71      ` + tests + `
72  }
73  `
74      return testJava
75  }
76  func createTestCsharp() string {···
121 [TestFixture]
122 public class CSharpTempTest
123 {` + tests + `
124 }`
125     return testCSharp
126 }
```

Código 2.28. Código para generar código de pruebas unitarias

Tanto el método `createTestJava` y como el método `createTestCShpar`, permiten generar un *string* de datos con la información de los casos de pruebas que está contenido en la variable `test`. Lo más importante que hay que destacar es que el resultado generado se lo devuelve como un *string* y la utilidad `iofile.go` que se encuentra dentro de la carpeta `utils`, se encargará de introducir dicho *string* en un archivo que estará almacenado en el espacio de trabajo de pruebas.

Como se puede visualizar en dicho código, hay texto que está al mismo nivel del nombre clave `func`. (líneas 69 a 73 y líneas 121 a 124). Esto se debe a que, el *string* generado, debe tener una identación correcta pues lo exige el compilador. Si no se establece identación en las líneas indicadas, el resultado es que el *string* devuelto, será guardado tal y como se ha generado; con espacios en blanco y una mala identación, generando errores de compilación.

La utilidad `iofile.go` contiene funciones que permiten realizar operaciones de creación de carpetas para los espacios de trabajo, creación de archivos que contendrán las pruebas unitarias y, creación de archivos que contendrán el código de los usuarios participantes.

En el Código 2.29, que está contenido dentro archivo `iofile.go`, se muestra la función que permite crear los archivos. La creación de archivos se ejecuta desde la línea 58 a la línea 73; primero, en la línea 58, se crea una carpeta, para ello, el método `Mkdir` de la

clase `os`, recibe dos parámetros: el primero corresponde a la ruta donde será creada la carpeta con el nombre del problema y, en el segundo parámetro se asignan los permisos de lectura, escritura y ejecución (`0777`).

A continuación, en el ciclo `for` se busca el lenguaje que está asociado a dicha prueba, para poder asignar el tipo de extensión que se agregará al archivo (líneas 60 a 70) y posteriormente, en la línea 71 se crea el archivo con la información obtenida de la ruta de la carpeta creada y el nombre del archivo con la extensión correspondiente (líneas 71 a 73). El archivo generado puede ser una de las siguientes opciones: `JavaTempTest.java` o `CSharpTempTest.cs`.

```
46  func (f File) ForProblem(nameProblem string) File {
47      (···
55      )
56      if err == nil {
57          if !status {
58              os.Mkdir(pathFolder, os.FileMode(0777))
59              for _, value := range f.Languages {
60                  if value == "Java" {
61                      name = value
62                      ext = "java"
63                  }
64                  if value == "CSharp" || value == "C#" {
65                      name = "CSharp"
66                      ext = "cs"
67                  }
68                  if f.NameFolder == "tests" {
69                      test = "Test"
70                  }
71                  file, _ := os.Create(pathFolder +
72                      "/" + name + "Temp" + test + "." + ext)
73                  defer file.Close()
74              }
75          }
76      }
77      return f
78  }
```

Código 2.29. Código para crear archivos

Finalmente, el archivo `test_formatter.go` contienen código que permite dar el formato correspondiente a la respuesta capturada desde la interfaz de línea de comando de Go.

El formato de respuesta se puede ver en las líneas de la 53 a la 58 del Código 2.30. Contiene únicamente la información relevante que la aplicación web necesita para procesarla. El CLI retorna un *string* con la información de la evaluación, y con el uso de

expresiones regulares (línea 42 y 43) se procede a buscar la información que es relevante.

```
31  func (dt dataTest) searchDataJava(data string) dataTest {
32
33      /*
34          when a code pass all tests, then data is like:
35          Time: 0,004, OK (3 tests)
36          else:
37          Time: 0,006, Tests run: 3,  Failures: 3
38      */
39      temp := strings.Split(data, ",")
40      if strings.Contains(data, "OK") {
41
42          regex := regexp.MustCompile("[0-9]+")
43          number := regex.FindAllString(temp[1], -1)
44
45          return map[string]string{
46              "NumberTests": number[0],
47              "Passed":      number[0],
48              "Failed":      "0",
49              "Duration":    strings.Split(temp[0], ":")[1],
50          }
51
52      }
53      return map[string]string{
54          "NumberTests": strings.Split(temp[1], ":")[1],
55          "Passed":      "0",
56          "Failed":      strings.Split(temp[2], ":")[1],
57          "Duration":    strings.Split(temp[0], ":")[1],
58      }
59  }
```

Código 2.30. Función para formatear el resultado devuelto por el CLI

b. Ejecución de pruebas unitarias para evaluar código

Se han considerado varios aspectos que son:

- Cada equipo tiene su propio espacio de trabajo segmentado únicamente por carpetas y no a nivel del lenguaje de programación ya se por paquetes para Java o espacios de nombres para C#.
- Los espacios de trabajo de cada equipo, tienen carpetas para cada problema que se va a resolver.
- Las pruebas unitarias no se escriben cada vez que se evalúa el código de cada equipo participante, sino que están centralizadas en su espacio de trabajo y son reutilizadas en cada ejecución de evaluación.
- Las pruebas unitarias están en su propio espacio de trabajo.

A continuación, en el Código 2.31, se muestra el *script* de evaluación de código Java. Primero, de la línea 2 a la línea 7, se declaran variables de entorno que serán utilizadas como rutas de acceso para cada clase para las pruebas unitarias y para las clases que se van a evaluar.

Con esto se evita establecer un único punto de acceso a los compiladores e intérpretes de Java y C# y, adicionalmente, se evita utilizar espacios de nombres para C# o paquetes para Java haciendo que la evaluación de código sea mucho más sencilla puesto que se buscan clases con nombres genéricos en carpetas de cada usuario.

De la línea 9 a la 11, se compila las pruebas unitarias generando un archivo con extensión `.class`. A continuación, se ejecuta la clase que contiene el código que fue enviado por el competidor (líneas 12 a 17). En este proceso de compilación se debe indicar la ruta del *test* que anteriormente fue compilado y la ruta de la clase que se va a probar.

Cuando se ejecuta la compilación y, posteriormente las pruebas, la librería de pruebas unitarias mostrará su propia estructura de texto del resultado obtenido. Previo a que se muestre en consola, se utiliza los `pipelines` de Linux, conjuntamente con el comando `grep`, para capturar dicha información, de tal manera que se pueda manipular y obtener solo lo que se requiere. Esta información es: el tiempo de ejecución, el número de pruebas ejecutadas, el tiempo de compilación y ejecución y si las pruebas pasaron o no.

```
1   #!/bin/bash
2   NAMEGROUP=$1,NAMEPROBLEM=$2
3   IRONCODER="/opt/ironcoder"
4   TESTS=$IRONCODER"/tests/"$NAMEPROBLEM
5   USERSPACE=$IRONCODER"/workspaces/"$NAMEGROUP"/"$NAMEPROBLEM
6   VENDOR=$IRONCODER"/vendor"
7   JAVATEMP="JavaTemp"
8   javac -d $USERSPACE $USERSPACE"/"$JAVATEMP".java"
9   javac -cp $USERSPACE:\
10      $TESTS:$VENDOR"/java/"junit.jar:$VENDOR"/java/"hamcrest.jar \
11      $TESTS"/"$JAVATEMP"Test.java"
12  java  -cp $USERSPACE:$TESTS:\
13            $VENDOR"/java/"junit.jar:$VENDOR"/java/"hamcrest.jar \
14            org.junit.runner.JUnitCore \
15            $JAVATEMP"Test" | grep \
16              -e "Time:*" -e "Tests run:*" \
17              -e "OK:*" | sed -e ':a;N;$!ba;s/\n/, /g'
18  find $USERSPACE $TESTS -name "*.class" -type f -delete
```

Código 2.31. Código para evaluar Java a través de *scripting* Bash

Al utilizar las variables de entorno como declaraciones del *script*, se evita crear paquetes para Java, en lo que se reduce la complejidad de buscar por paquetes el *script* deseado.

Para ello también se ha definido que el nombre del archivo que contiene la solución al problema planteado, tenga como nombre de clase `JavaTemp`. Así mismo con el archivo de pruebas, su nombre de clase será `JavaTempTest`. Con esta consideración, se reutiliza el código de las pruebas y únicamente se pasa como referencia, en la ejecución de compilación y pruebas.

El mismo proceso se sigue para ejecutar las pruebas sobre CSharp. El *script* para C# se muestra en el Anexo I.

2.4 Despliegue

Previo al paso de despliegue del prototipo, es importante definir el *script* de los archivos `Dockerfile` que serán utilizados para desplegar la aplicación web y los servicios web en el servicio Elastic BeanStalk de AWS. Es importante mencionar que el servicio S3 es usado para descargar la llave de seguridad que permitirá conectarse con GitLab y descargar el código fuente de tanto de la aplicación web, como del servicio web, en el proceso de creación de los contenedores.

2.4.1. Archivo Dockerfile para la aplicación web

Cuando se construye una imagen base, Docker creará contenedores temporales por cada sentencia que se ejecute y que haya sido especificada en el archivo `Dockerfile`. Esto hará que la imagen resultante tenga varias capas resultando también en un mayor tamaño de dicha imagen.

Para reducir al máximo el número de capas que se generan y así también el tamaño de la imagen resultante, se utiliza una única sentencia `RUN` en el archivo `Dockerfile` y, los *scripts* del *shell* serán ejecutados uno a continuación de otro. Esto permitirá que el tamaño final de la imagen sea realmente pequeño y que, para el proceso de despliegue, los tiempos también se reduzcan.

A continuación, se describe la ejecución del *script* del Código 2.32. En la línea 1 conjuntamente con la línea 2 y la línea 3, se crea la carpeta `www` en la ruta `/go/src`, y se le asignan los permisos correspondientes. La línea 5, a través del comando `apk add` de Alpine, descargará todas las actualizaciones del sistema operativo base y dependencias como `bash`, `git`, y `openssh-client`. En la línea 4, se descarga la llave SSH, que está alojada en el servicio S3 de AWS, utilizando el comando `curl`. Esta llave se agrega en la ruta temporal `/tmp/id_rsa` para posteriormente establecer los permisos correspondientes (línea 8). En la línea 9, se ejecuta el agente SSH y adicionalmente, el

resultado devuelto de la ejecución se evalúa para verificar si es que el agente SSH ha empezado a ejecutarse normalmente.

```
1   RUN mkdir -p \
2       /go/src/www \
3       && chmod 755 -R /go/src/www \
4       && echo "void" \
5       && apk add --update --no-cache curl bash git openssh-client \
6       && curl -0 https://s3-sa-east-1.amazonaws.com/private-rsa/id_rsa \
7       && -o /tmp/id_rsa \
8       && chmod 600 /tmp/id_rsa \
9       && eval $(ssh-agent) \
10      && echo -e "StrictHostKeyChecking no" >> /etc/ssh/ssh_config \
11      && ssh-add -k /tmp/id_rsa \
12      && git clone git@gitlab.com:ironcoder/www.git /go/src/www \
13      && cd /go/src/www && go get -u github.com/golang/dep/cmd/dep \
14      && go get github.com/codegangsta/gin \
15      && dep ensure \
16      && ln -s /go/src/www/run.sh /usr/bin/runwww \
17      && rm -rf tmp/*
18  WORKDIR /go/src/www
19  ENTRYPOINT [ "runwww" ]
20  EXPOSE 3000
```

Código 2.32. Dockerfile para construir imagen de la aplicación web.

En la línea 10 se agrega en el archivo de configuración para SSH, el parámetro `StrictHostKeyChecking` establecido en `no` de tal manera que no se valide de forma estricta el proceso de verificación de credenciales cuando se proceda a conectarse con el repositorio de GitLab, para descargar los proyectos (línea 12).Las llaves SSH que previamente fueron descargadas, se agregan al agente SSH (línea 11). Ahora ya es posible clonar el repositorio de la aplicación web utilizando el comando de la línea 12.

Después de descargar la aplicación web, se procede a ejecutar el comando de la línea 13 y 14, para descargar las dependencias globales de Go y, en la línea 15, se ejecuta el comando para que sean descargadas las dependencias del proyecto.

Finalizado el proceso de descarga de dependencias, se crea un enlace simbólico del *script* de ejecución del servidor web, con el propósito de que este servidor sea ejecutado de forma automática, cuando el contenedor Docker comience su ejecución (línea 16).

En la línea 18 se establece el espacio de trabajo donde debe iniciar el contenedor Docker. En la línea 19 se especifica que comando debe ejecutarse, el mismo que fue creado en la línea 16. Finalmente, se expone el puerto por el cual se escucharán las

peticiones. Es importante mencionar que la exposición de este puerto es para que otros servicios puedan acceder al contenedor, sin embargo, el acceso al servicio o a la aplicación será generalmente por el puerto 80 para HTTP.

2.4.2. Archivo Dockerfile para el servicio web

El servicio web se encarga de ejecutar tareas como compilación y evaluación de código, además de crear los espacios de trabajo y crear archivos que contiene el código fuente ya sea de las pruebas unitarias que se crean o de los algoritmos que son enviados por los usuarios.

Ahora se deben tener en consideración varios detalles que tienen que ver no solo con las dependencias que deben ser descargadas sino también, con los espacios de trabajo que deben ser creados.

El Código 2.33, corresponde al archivo `Dockerfile` para construir el servicio web sobre una imagen de Docker.

De la línea 1 a la línea 10, se crean todas las carpetas que serán utilizadas por el servicio web y se añade los permisos correspondientes.

Se ha definido la ruta `/opt/ironcoder` para el espacio de trabajo donde se guardarán los *scripts* de las pruebas unitarias, la creación de carpetas para cada usuario, y la creación de carpetas para cada problema planteado.

De la línea 11 a la línea 17, se descargan todas las dependencias que corresponden al *runtime* Mono para C#, el JDK para Java, `curl, git, open-ssh` (línea 11 a 13), la librería de pruebas unitarias JUnit (línea 15 a 17), el gestor de paquetes `nuget` para C# que permitirá descargar la librería de pruebas unitarias NUnit (línea 28 y 29).

El proceso de clonar el repositorio donde está alojado el servicio web es el mismo que se emplea en la clonación de la aplicación web.

De la línea 18 a la 24, se descarga el código fuente del servicio web. Se crea un enlace simbólico del acceso al main de la aplicación (línea 30) para que esta pueda ser ejecutada cuando el contenedor Docker empiece su ejecución.

La línea 31 permite borrar todas las dependencias que fueron descargadas para Alpine. De la línea 32 a la 38 se borran todos los archivos y carpetas que ya no son útiles.

Este proceso de quitar herramientas, eliminar archivos y carpetas se da por el hecho de que se desea mantener un contenedor limpio y lo más pequeño posible además de que

también se logra reducir en un pequeño porcentaje el consumo de recursos de memoria y procesamiento del servicio Elastic BeanStalk de AWS que se está utilizando para desplegar el servicio web con el propósito de que no se exceda en el límite de consumo de dicho servicio y así evitar generar costos económicos.

```
RUN mkdir -p \
    /var/www \
    /opt/ironcoder \
    /opt/ironcoder/scripts \
    /opt/ironcoder/tests \
    /opt/ironcoder/vendor \
    /opt/ironcoder/vendor/java \
    /opt/ironcoder/vendor/dotnet \
    /opt/ironcoder/workspaces \
    && chmod 755 -R /opt/ironcoder/workspaces /opt/ironcoder/tests /opt/ironcoder/scripts \
    && echo "@testing http://dl-4.alpinelinux.org/alpine/edge/testing" >> /etc/apk/repositories \
    && apk add --update --no-cache mono@testing mono-dev@testing openjdk8 curl git openssh-client \
    && curl -O https://dist.nuget.org/win-x86-commandline/latest/nuget.exe \
    && curl -0 https://repo1.maven.org/maven2/junit/junit/4.12/junit-4.12.jar \
    -o /opt/ironcoder/vendor/java/junit.jar \
    && curl -0 https://repo1.maven.org/maven2/org/hamcrest/hamcrest-core/1.3/hamcrest-core-1.3.jar
    -o /opt/ironcoder/vendor/java/hamcrest.jar \
    && curl -0 https://s3-sa-east-1.amazonaws.com/private-rsa/id_rsa -o /tmp/id_rsa \
    && chmod 600 /tmp/id_rsa \
    && eval $(ssh-agent) \
    && echo -e "StrictHostKeyChecking no" >> /etc/ssh/ssh_config \
    && ssh-add -k /tmp/id_rsa \
    && git clone git@gitlab.com:ironcoder/binary-webservice.git /var/www \
    && git clone git@gitlab.com:ironcoder/scripts.git /opt/ironcoder/scripts \
    && rm -rf /etc/ssl/certs/java \
    && cat /etc/ssl/certs/* >ca-bundle.crt \
    && cert-sync ca-bundle.crt \
    && mono nuget.exe install NUnit -Version 3.9.0 -OutputDirectory "/opt/ironcoder/vendor/dotnet"
    && mono nuget.exe install NUnit.Runners -Version 3.7.0 -OutputDirectory "/opt/ironcoder/vendor/
    && ln -s /usr/lib/jvm/java-1.8-openjdk/bin/javac /usr/bin/javac \
    && apk del curl git bash openssh-client tar musl musl-utils \
    && rm ca-bundle.crt \
    && rm -rf /var/www/.git \
    && rm -rf /var/cache/apk/* \
    && rm -rf tmp/* \
    && rm -rf nuget.exe \
    && rm -rf *.tar.gz \
    && rm -rf *.jar
EXPOSE 4000 4001
ENTRYPOINT ["/var/www/main"]
```

Código 2.33. Dockerfile para construir imagen del servicio web

2.4.3. Creación de servicios en AWS

A continuación, se muestra el proceso de despliegue de los servicios en AWS para desplegar el prototipo propuesto.

En primer lugar, se debe crear la base de datos utilizando el servicio RDS (*Relational Database Service*) de AWS, pues es necesario obtener los parámetros de configuración para que la aplicación web pueda ser ejecutada. Luego, se debe crear el servicio web utilizando el servicio Elastic BeanStalk (EBS) de AWS. Finalmente, al crear un servicio

de Elastic BeanStalk para ejecutar la aplicación web será requerida. El proceso de creación del servicio para la aplicación web es el mismo que se presenta para crear el servicio de EBS para el servicio web.

a. Lanzamiento del servicio de base de datos utilizando el servicio RDS

Las características del servicio de base de datos del cual se va a disponer se presentan en la Tabla 2.2. Estas características son las mínimas que provee AWS y son ofrecidas como un servicio gratuito siempre y cuando no se superen los límites de uso.

Tabla 2.2. Características del servicio de base de datos RDS de AWS

Instancia DB	
Tipo	db.t2.micro
CPU Virtual	1 vCPU
Memoria RAM	1Gib
Free Tier Eligible	Yes
Version	MySQL 5.7.21
Storage	20 GiB

A continuación, se describe el proceso de creación del servicio RDS de AWS.

1. Habiendo iniciado sesión en una cuenta en AWS, y, posteriormente, seleccionando el servicio RDS. Se mostrará una sección como se ve en la Figura 2.53. Como se puede observar, existen dos opciones, en este caso, se presiona el botón *Launch a DB instance.*

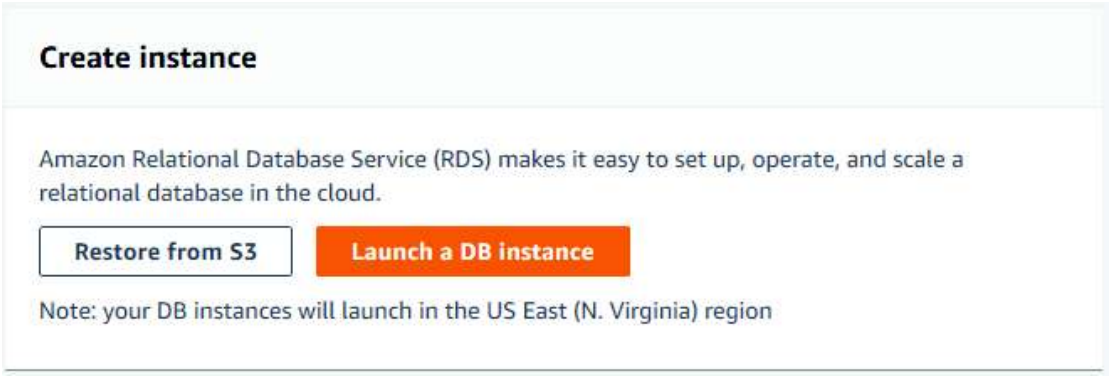


Figura 2.59. AWS - crear servicio RDS

2. Se escoge MySQL y se marca la opción "*only enable options eligible for RDS Free Usage Tier*" para solo obtener las configuraciones gratuitas (Figura 2.54). Estas configuraciones permitirán que el uso de los recursos de RDS se puedan establecer de forma automática a los valores más bajos y que son utilizados solo para pruebas.

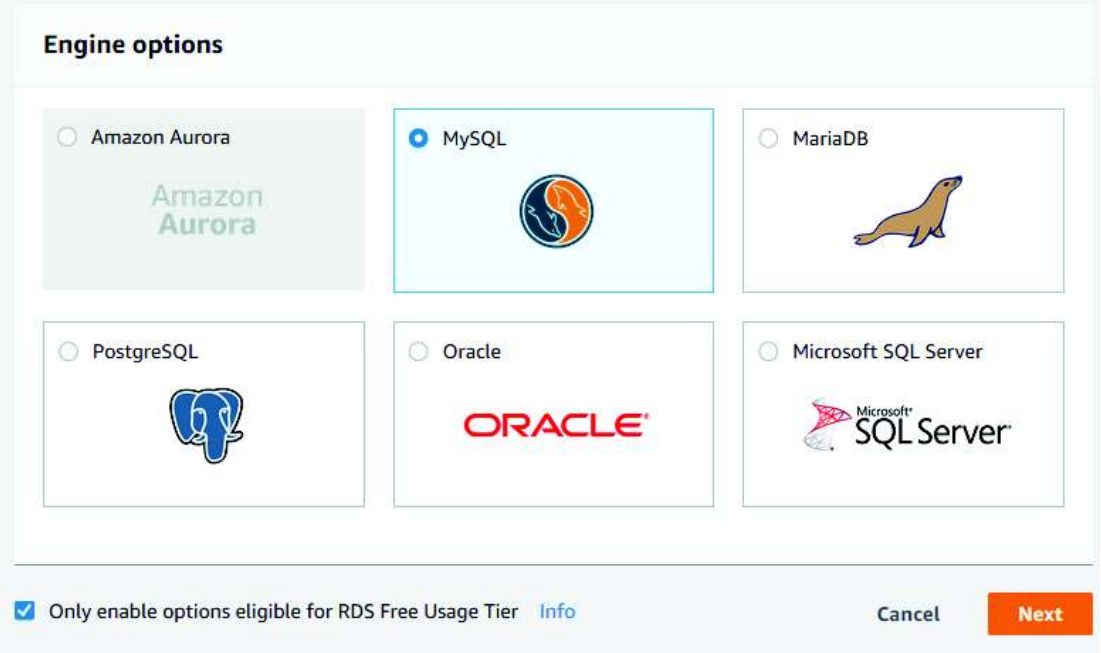


Figura 2.60. AWS - seleccionar base de datos

3. Se selecciona la versión 5.7.21 de MySQL pues soporta el almacenamiento de datos de objetos JSON, que se utiliza em este prototipo (ver Figura 2.61).

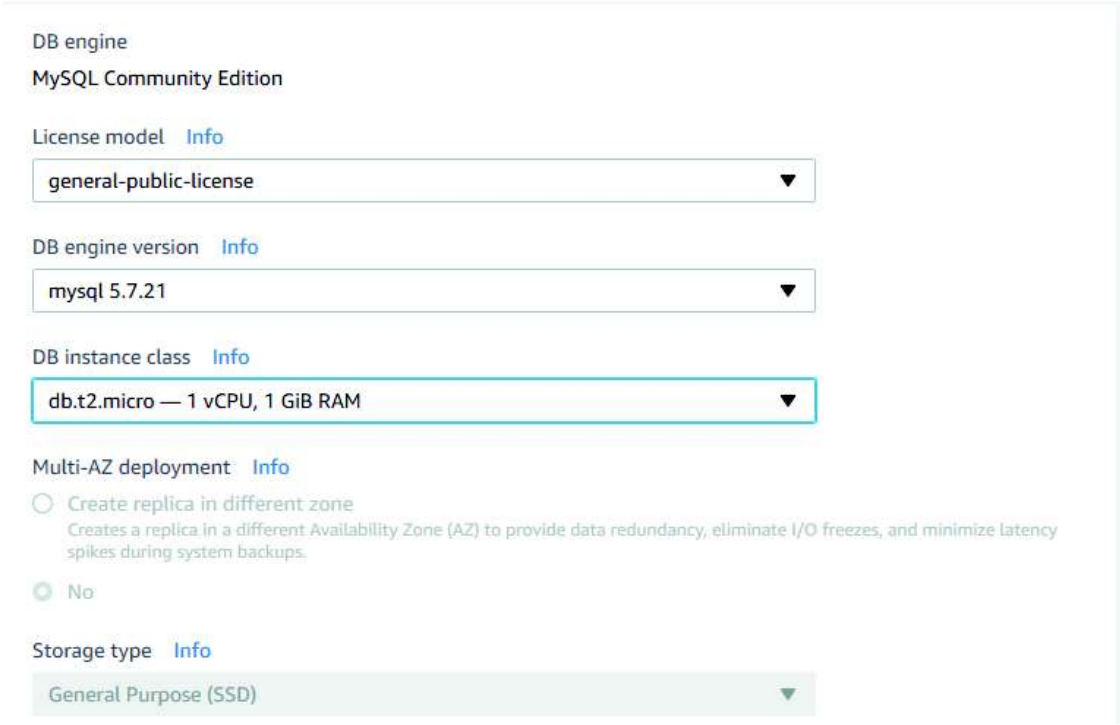


Figura 2.61. AWS – seleccionar versión de MySQL

4. En esta etapa, cómo se ve en la Figura 2.62 y la Tabla 2.3, se deben establecer los parámetros de conexión a la base de datos.

Tabla 2.3. Parámetros de configuración para RDS

DB Instance Identifier	tesis
Master username	ironcoder
Master password	ironcoder

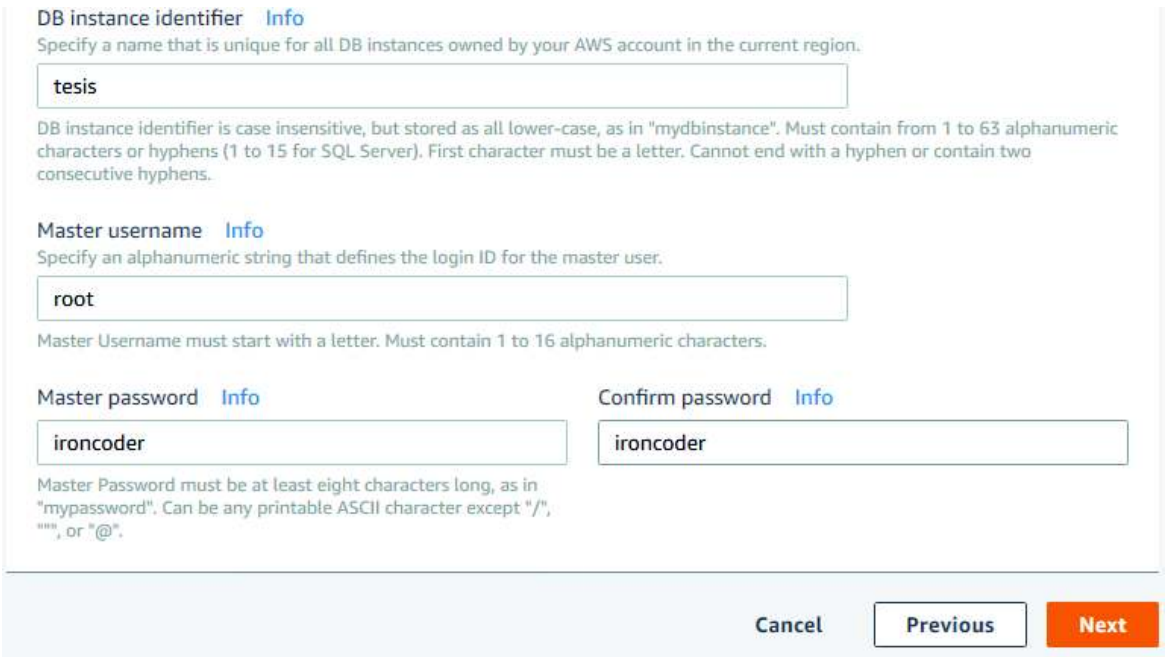


Figura 2.62. AWS - configurar datos de conexión de la base de datos

5. A continuación, se asigna un nombre a la base de datos y un puerto. Las demás opciones quedan tal y como están (ver Figura 2.63).

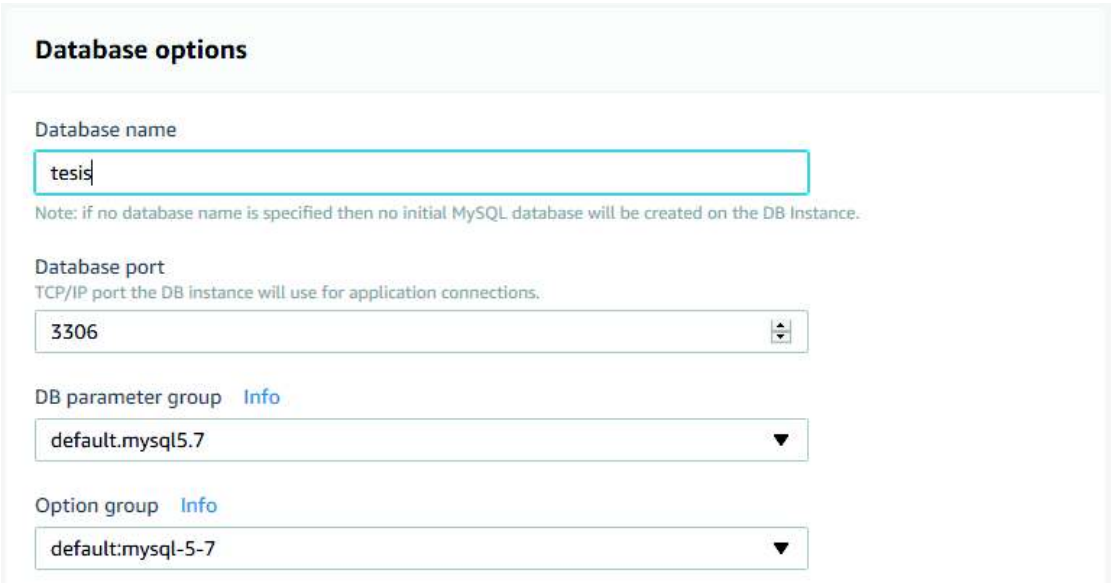


Figura 2.63. AWS – configurar nombre y puerto de la base de datos

6. En esta etapa no hay nada que configurar, se dejan todas las configuraciones por defecto y al final de formulario se mostrará una sección como se ve en la Figura 2.64. Se presiona el botón *Launch DB instance*, para crear la instancia del servicio de base de datos de RDS.

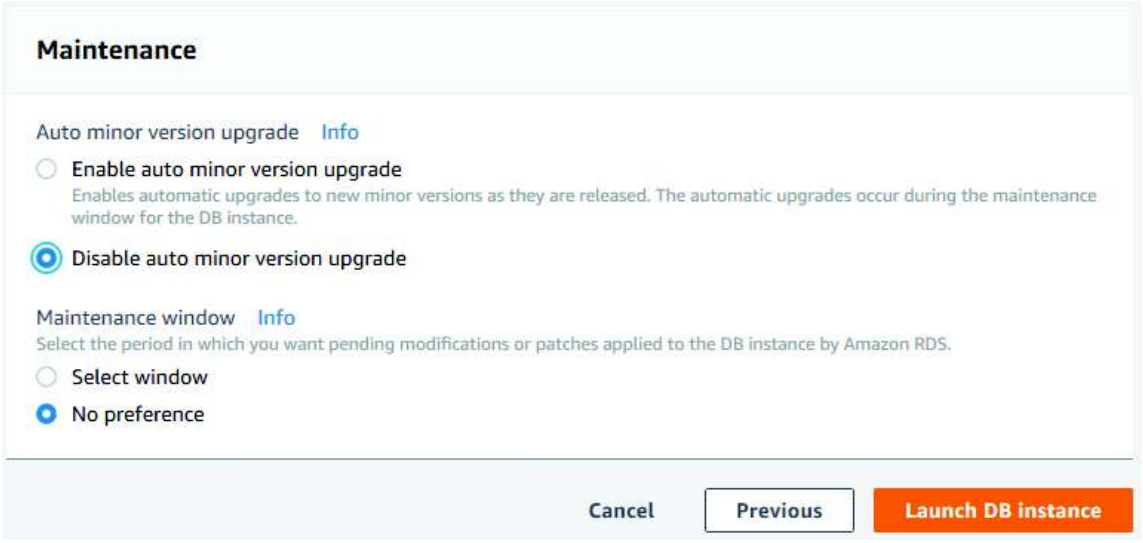


Figura 2.64. AWS – ejecución de la instancia de base de datos

7. Finalmente, se lanzará el servicio mostrando un mensaje como se ve en la Figura 2.65. Para obtener la URL de acceso a la instancia que se ha creado, para que pueda ser utilizada tanto por la aplicación web como por cualquier cliente MySQL, se procede a presionar el botón *View DB instance details.*

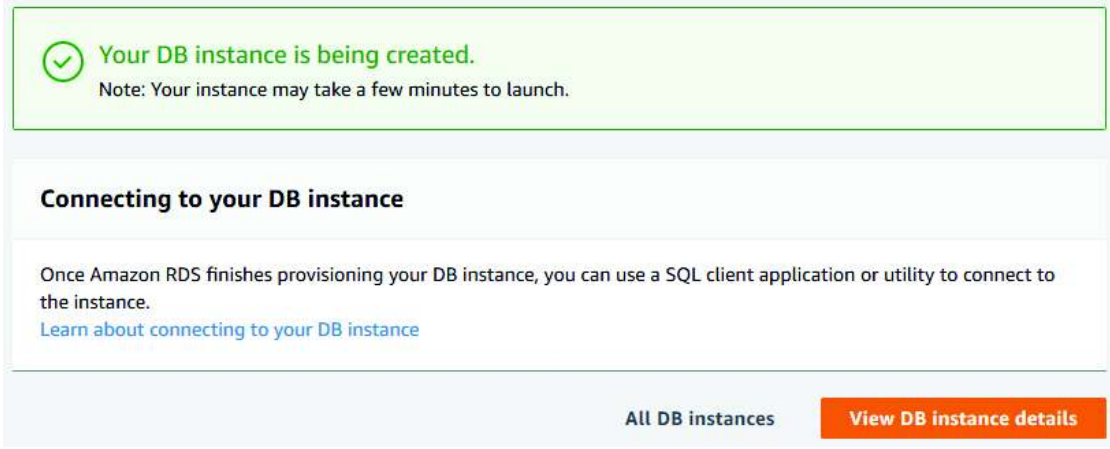


Figura 2.65. AWS – instancia RDS creada

Se mostrarán varias secciones con información acerca de la instancia que ha sido creada. Se busca la sección `connect` para buscar la URL de conexión. Ésta se encuentra debajo del título `Endpoint` como se muestra en la Figura 2.66.

Figura 2.66. Obteniendo la URL de acceso al servicio RDS

b. <u>Desplegue del servicio web utilizando Elastick BeanStalck (EBS)</u>

El proceso que a continuación se describe, se utilizará también para crear la instancia para la aplicación web, solo se deben considerar las variables de entorno de la Tabla 2.4.

1. Se selecciona el servicio de Elastic Beanstalk y a continuación se presiona el botón *Create New Application*. Aparecerá un cuadro de diálogo como se muestra en la Figura 2.62.

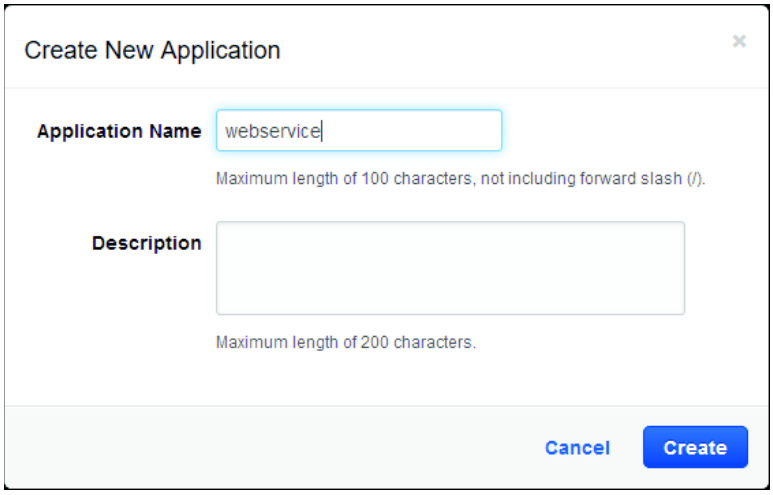


Figura 2.67. AWS – creación de una aplicación en EBS

2. A continuación, se presenta un panel como se muestra en la Figura 2.68. Para crear un ambiente de desarrollo para la aplicación, se presiona el *link Create one now* y se redireccionará a una nueva página.

Figura 2.68. AWS – creación de ambiente de trabajo en EBS

3. Ahora se debe seleccionar el tipo de ambiente que va a ser desplegado. Se selecciona la opción *Web server enviroment* (ver Figura 2.69). A continuación, se presiona el botón *select*.

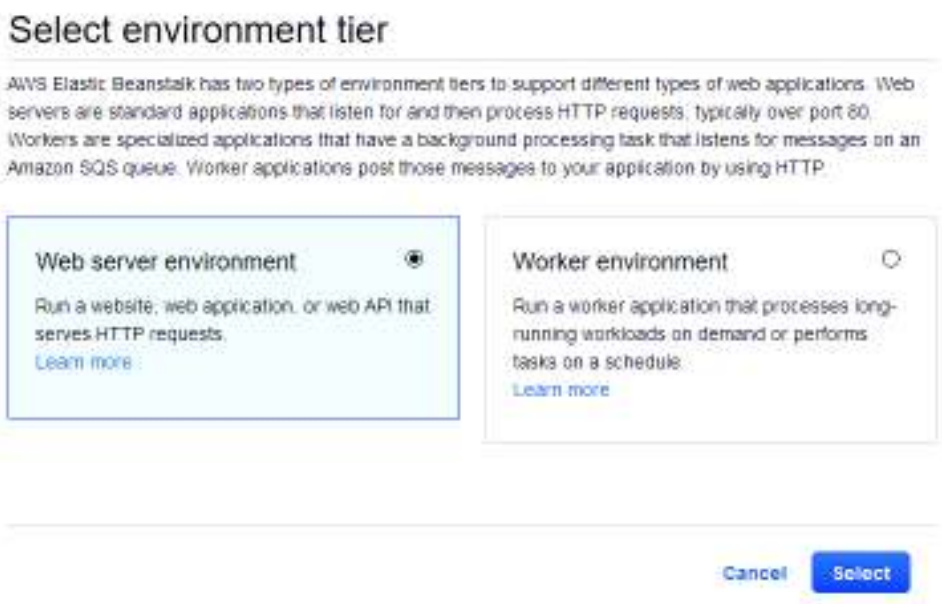


Figura 2.69. AWS – seleccionar el ambiente *web server environment*

4. En esta sección se verifica que el subdominio utilizado para el servicio web, esté disponible (ver Figura 2.70).

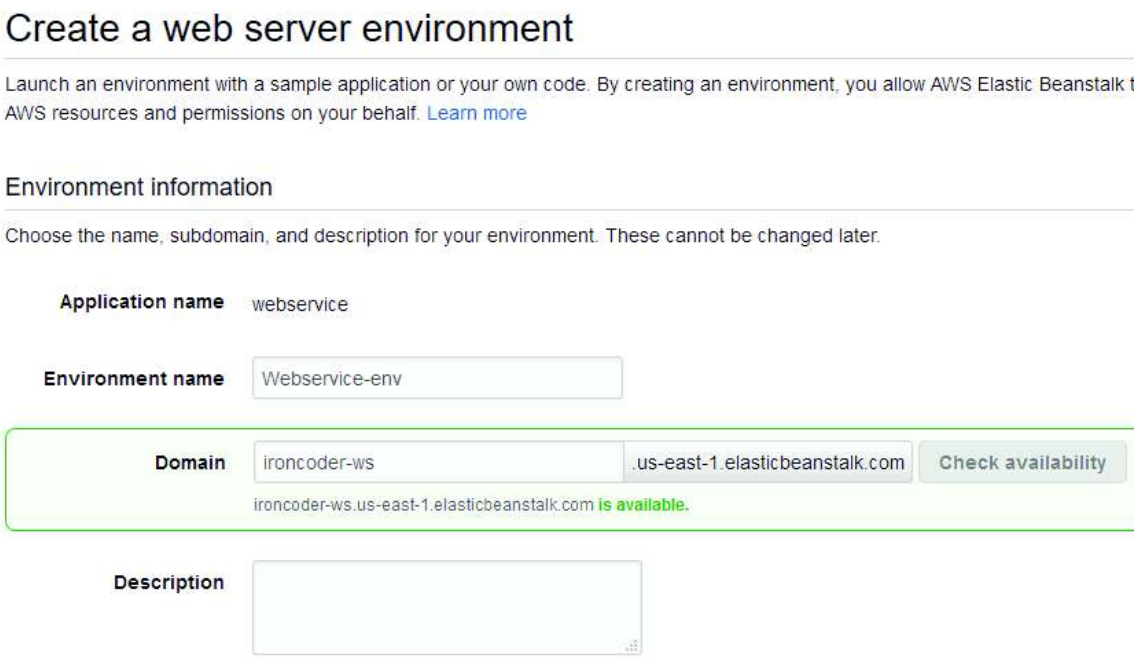


Figura 2.70. AWS – verificacar disponibilidad del sub-dominio

5. A continuación, se selecciona la plataforma preconfigurada Docker y se marca la opción *Upload your code*, se abrirá una nueva pestaña para cargar el archivo `Dockerfile`. Luego, se presiona el botón *Configure more options* y se mostrará un nuevo panel. Se debe buscar la sección Software (ver Figura 2.71).

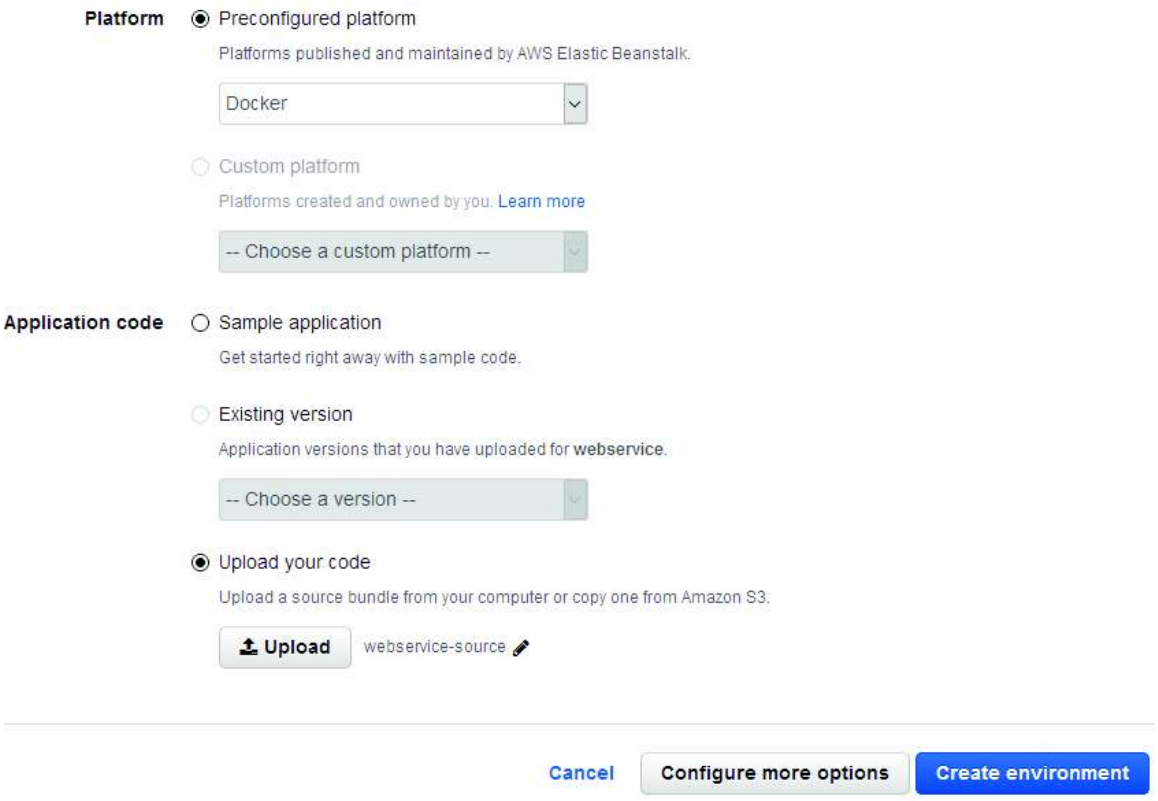


Figura 2.71. AWS - seleccionar Docker y subir `Dockerfile`.

6. Se procede a presionar *modify* (ver Figura 2.72), e ir a la sección de variables de entorno (ver Figura 2.73).

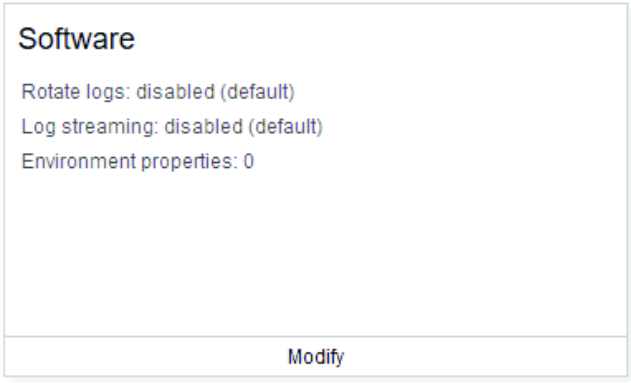


Figura 2.72. AWS – acceso a variables de entorno

7. Se agregan las variables de entorno en la sección correspondiente (Figura 2.73)

Figura 2.73. AWS – agregando variables de entorno

Las variables de entorno que están asociadas al servicio web y que deben ser agregadas al ambiente que se está creando se muestra en la Tabla 2.4.

Tabla 2.4. Variables de entorno del servicio web

Nombre	Valor
URL	0.0.0.0
PORT	80
ID	IronCoderWS
IRONCODER_TOKEN	61fbc82d-bf35-4742-886f-74652e63a951

Se presiona el botón *save* el cual se retornará a una página anterior. A continuación, al final de la página se presiona el botón *create enviroment* y la instancia empezará a ser creada. Se mostrará nueva página como se ve en la Figura 2.74.

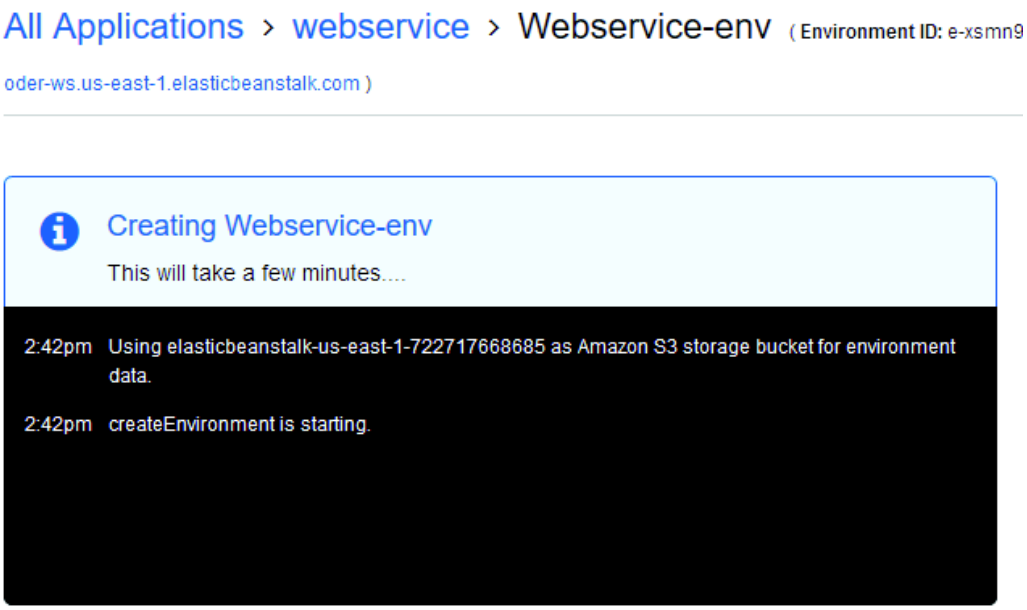


Figura 2.74. AWS – creación del servicio web en la instancia en EBS

Para crear la instancia de la aplicación web se siguen los mismos pasos que se han considerado para crear el servicio web, pero las variables de entorno que deberán ser creadas se muestran en la Tabla 2.5.

Tabla 2.5 Variables de entorno de la aplicación web

Nombre	Valor
WEBSERVICE1_PROTOCOL	http
WEBSERVICE1_HOST	ironcoder-ws.us-east-1.elasticbeanstalk.com
WEBSERVICE1_PORT	80
WEBSERVICE1_TOKEN	61fbc82d-bf35-4742-886f-74652e63a951
WEB_HOST	ironcoder-wa.us-east-1.elasticbeanstalk.com
WEB_PORT	3000
ENVIROMENT	production
MYSQL_USERNAME	root
MYSQL_PASSWORD	ironcoder
MYSQL_DATABASE	tesis
MYSQL_HOST	tesis.c91wigahur9x.us-east-1.rds.amazonaws.com
MYSQL_PORT	3306
PASS_SMTP	
PORT_SMTP=587	587
SERVER_SMTP	smtp.gmail.com
FROM_SMTP	

3 RESULTADOS Y DISCUSIÓN

En esta sección se procede a realizar las pruebas correspondientes sobre el sistema que ya ha sido desplegado en los servicios de AWS. Las capturas de pantalla de las tablas de la base de datos se adjuntan en el Anexo V. La Tabla 3.1 y la Tabla 3.2 muestran los datos de conexión que se obtuvieron en la creación de los servicios en AWS.

Tabla 3.1. Datos de conexión a la base de datos de AWS

Datos de conexión de la base de datos	
Name database	tesis
Username	ironcoder
Password	ironcoder
Endpoint de acceso a la DB	tesis.c91wigahur9x.us-east-1.rds.amazonaws.com
Port	3306

Tabla 3.2. URL a los servicios web y a la aplicación web

Nombre	URL
Aplicación web	ironcoder-wa.us-east-1.elasticbeanstalk.com
Servicio web 1	ironcoder-ws.us-east-1.elasticbeanstalk.com
Servicio web 2	wwwservice-1.us-east-1.elasticbeanstalk.com

3.1 Pruebas de funcionamiento sobre la aplicación web

Se puede verificar que las instancias se han creado correctamente desde el panel de administración de AWS. Tanto para los servicios web como para la aplicación web, se mostrará una página como se ve en la Figura 3.1.

En la Figura 3.2 se visualiza el panel de administración del servicio Elastic BeanStalk. Se han creado dos instancias de servicios web y una instancia para la aplicación web. La instancia `wwwservice-1` fue desplegada a partir de la clonación de la instancia `wwwservice`.

Como se puede ver en la Figura 3.2, los servicios web se han desplegado correctamente, pues los bloques de información de cada servicio están de color verde. Cuando una instancia tiene problemas, el color utilizado será naranja. Y si, hay errores en el sistema que impiden que la instancia de EBS se ejecute normalmente y genere exceso de consumo de recursos, EBS advertirá marcando los bloques informativos de color rojo.

También se puede acceder a los servicios web o a la aplicación web, a través de las URL mostradas en la Tabla 3.2.

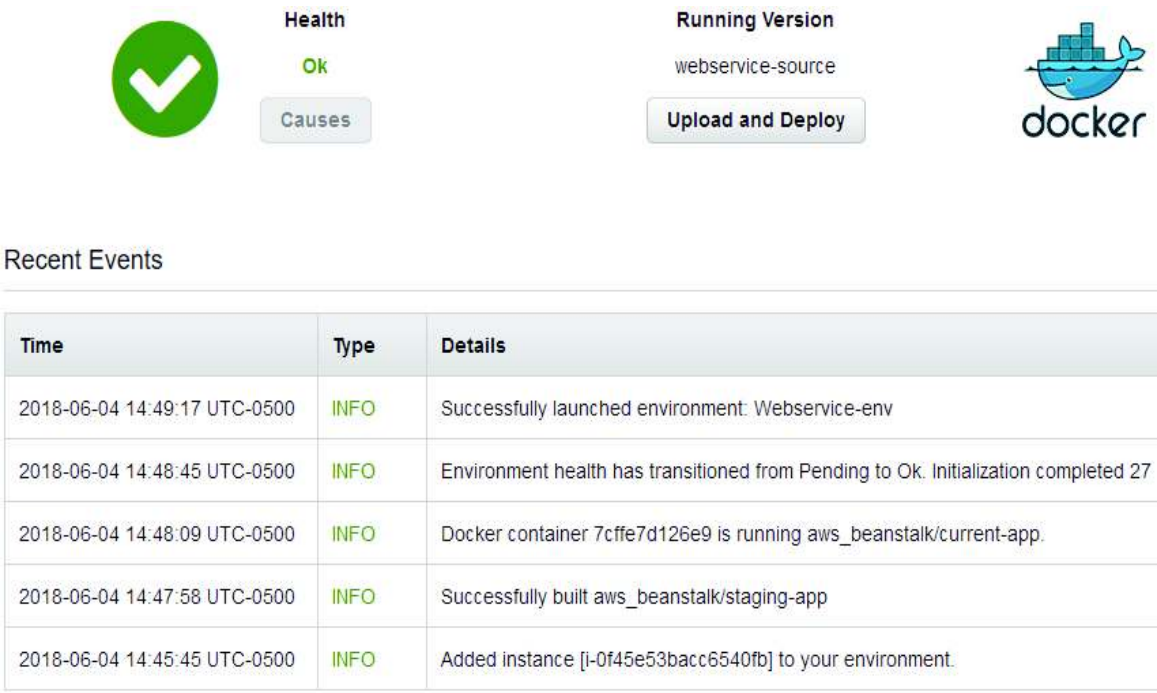


Figura 3.1. Estado del servicio web creado en Elastick BeanStalk

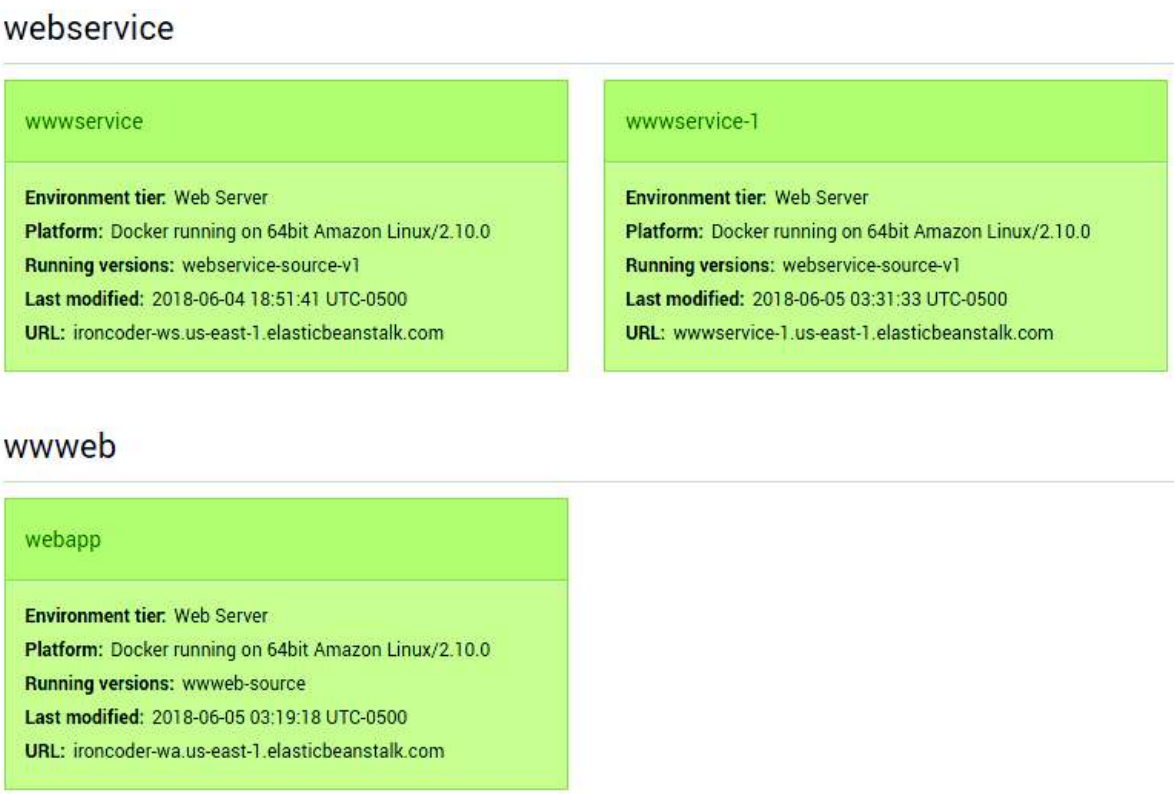


Figura 3.2. Servicios web y aplicación web ejecutándose correctamente.

El servicio web tiene un mecanismo de verificación externo a través del segmento de acceso a una URL. El servidor responderá con un objeto JSON y el estado de código 200 en el caso de los servicios web (ver Figura 3.3). En el caso de la aplicación, se podrá visualizar la pantalla inicial (ver Figura 3.4).

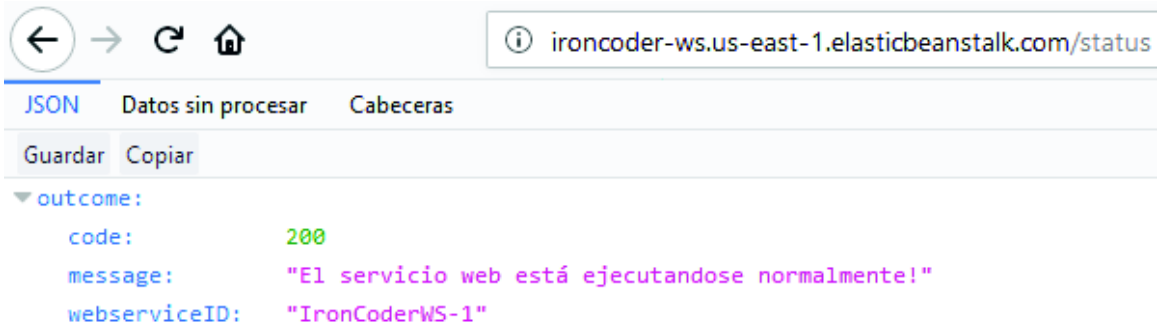


Figura 3.3. Comprobando el estado del servicio web a través de la URL

Cuando se accede a la aplicación web, se puede inferir que el servicio web también se encuentra trabajando normalmente. Es importante mencionar este detalle puesto que tal y como está construido el sistema, en el proceso de inicialización del servidor de la aplicación web, se comprueba que al menos un servicio web esté disponible.

Figura 3.4. Página de inicio de la aplicación web

En la Figura 3.5, se puede observar el formulario de inicio de sesión resultante. Se comprueba que la validación esté trabajando correctamente. Para ello, se ingresa cualquier dato que no tenga el formato de correo electrónico y se ingresa cualquier dato en el *input* para la contraseña. A continuación, se presiona el botón "*Iniciar sesión*" y el mensaje de advertencia será como se muestra en la Figura 3.6.

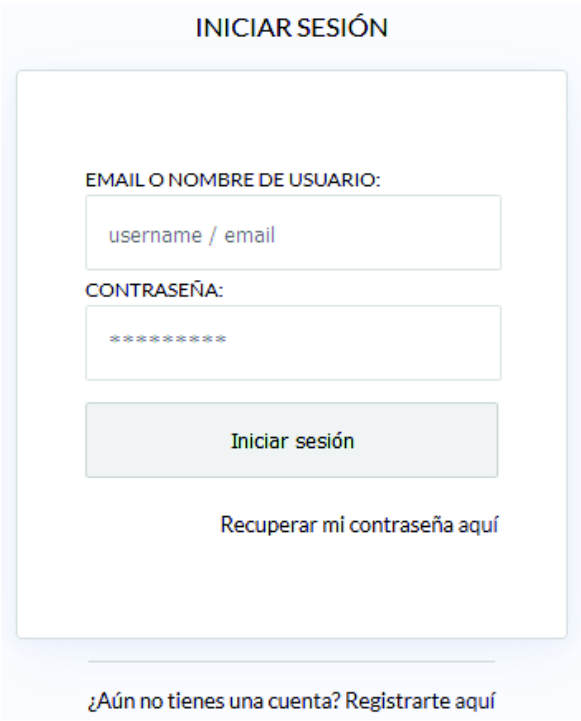


Figura 3.5. Formulario de inicio de sesión

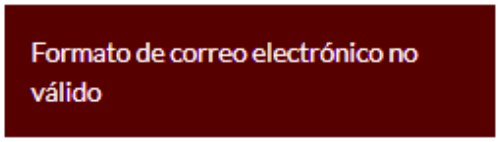


Figura 3.6. Mensaje de error de formato de correo no válido

También se puede comprar si el correo especificado ya está registrado en el sistema. El mensaje que se mostrará se puede observar en la Figura 3.7.

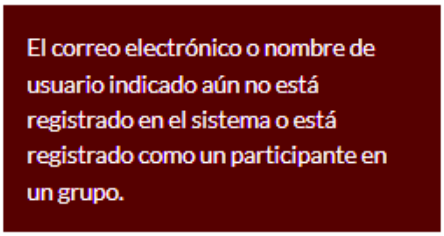


Figura 3.7. Mensaje de error de usuarios no registrados en el sistema

Las mismas pruebas se pueden realizar sobre los formularios de registro de usuarios y recuperación de contraseña. Se pueden ver en la Figura 3.8 y la Figura 3.9 respectivamente, las interfaces de usuario de ambos formularios.

Figura 3.8. Registro de usuarios

Figura 3.9. Recuperación de contraseña

Para iniciar sesión en el panel de administración, se procede a ingresar las credenciales `admin` para el nombre de usuario y `admin` para la contraseña.

El panel de administración resultante se muestra en la Figura 3.10.

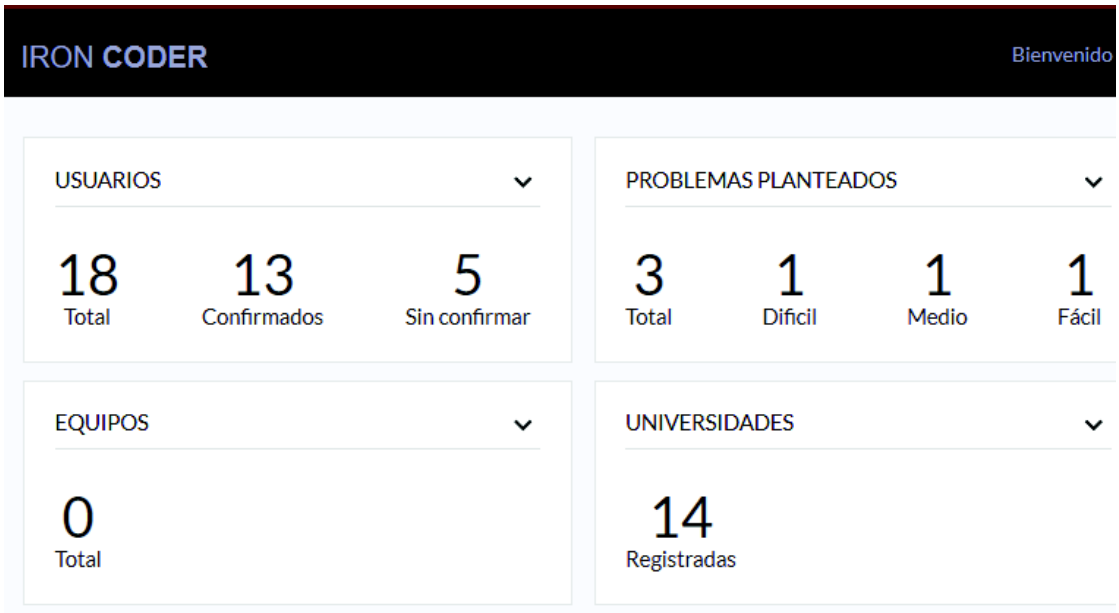


Figura 3.10. Panel de administración

Al presionar en el icono del extremo superior derecho de cada bloque, se desplegará una lista de opciones, como se muestra en la Figura 3.11. Al presionar en la opción “ver lista”, en cualquiera de las listas de los bloques del panel de administración, se mostrará la siguiente tabla. Es importante mencionar que dicha tabla es común para todas las secciones que tienen listas para mostrar.

El resultado se puede visualizar en la Figura 3.12.

Figura 3.11. Lista de opciones

Figura 3.12. Lista genérica para mostrar información de sistema

La página de notificación cuando un usuario cierra sesión se muestra en la Figura 3.13.

Figura 3.13 Página de notificación de cierre de sesión

La página de notificación cuando se desea acceder a un recurso que no existe se muestra en la Figura 3.14.

Figura 3.14 Página de error 404

La página que muestra el sistema de *ranking* de competidores se muestra en la Figura 3.15. En esta página, los 3 primeros competidores son colocados en la parte superior en secciones de bloques mucho más atractivos y mejor diseñados.

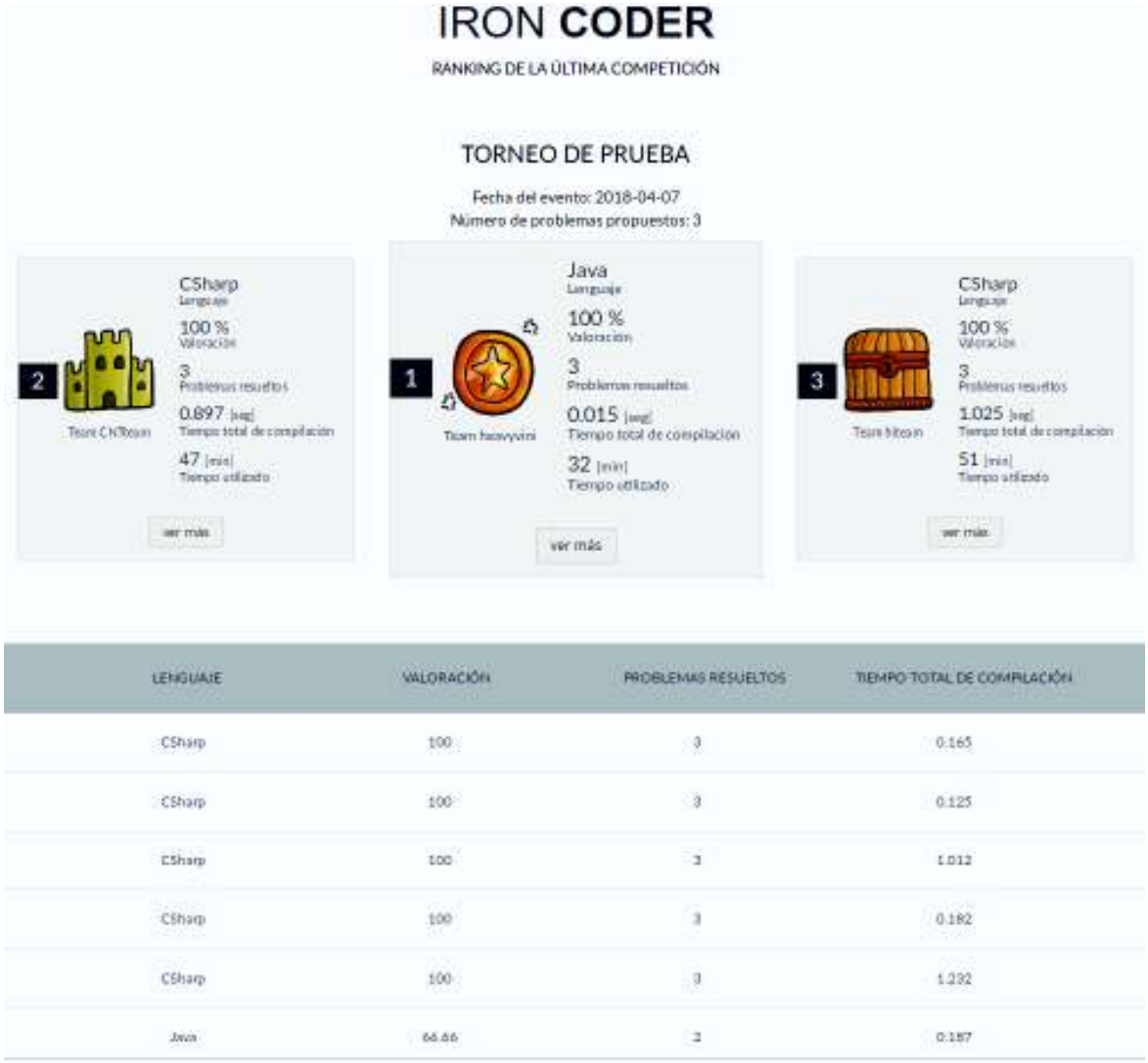


Figura 3.15. *Ranking* de participantes

En la Figura 3.16, se muestra el formulario que el usuario registrado deberá llenar para poder crear un equipo. En este proceso, el usuario puede seleccionar el lenguaje de programación y realizar la invitación a otros usuarios a formar parte de equipo vía correo electrónico.

Figura 3.16. Formulario para registrar un nuevo equipo

La Figura 3.17 corresponde al formulario que el usuario debe llenar para formar parte de un equipo.

En este formulario, el usuario que ha decidido participar en un torneo, podrá crear un equipo agregando un nombre para el equipo, el lenguaje que desea utilizar en la competición y finalmente realizar una invitación a través de correo electrónico a otros participantes. El número de *inputs* que el usuario verá para colocar correos electrónicos queda determinado por la definición que da el administrador de cuantos participantes podrán formar parte de un equipo.

Es importante mencionar que, para cada torneo, el usuario registrado podrá formar un nuevo equipo, seleccionar un nuevo lenguaje de programación y realizar la invitación a nuevos participantes.

Figura 3.17. Formulario para usuarios que formarán parte de un equipo

En la Figura 3.18, se muestra una lista de los correos que han sido enviados a los competidores. Además, el formato del correo electrónico que se envía a los participantes.

3.2 Ejecución de pruebas sobre el servicio web

Para realizar algunas pruebas sobre el servicio web ya sea para procesos de crear espacios de trabajo, así como los archivos que contienen código de Java o C#, tanto para

las pruebas unitarias, así como para crear los archivos que contiene el código que es enviado por los participantes, se tiene en consideración la siguiente información.

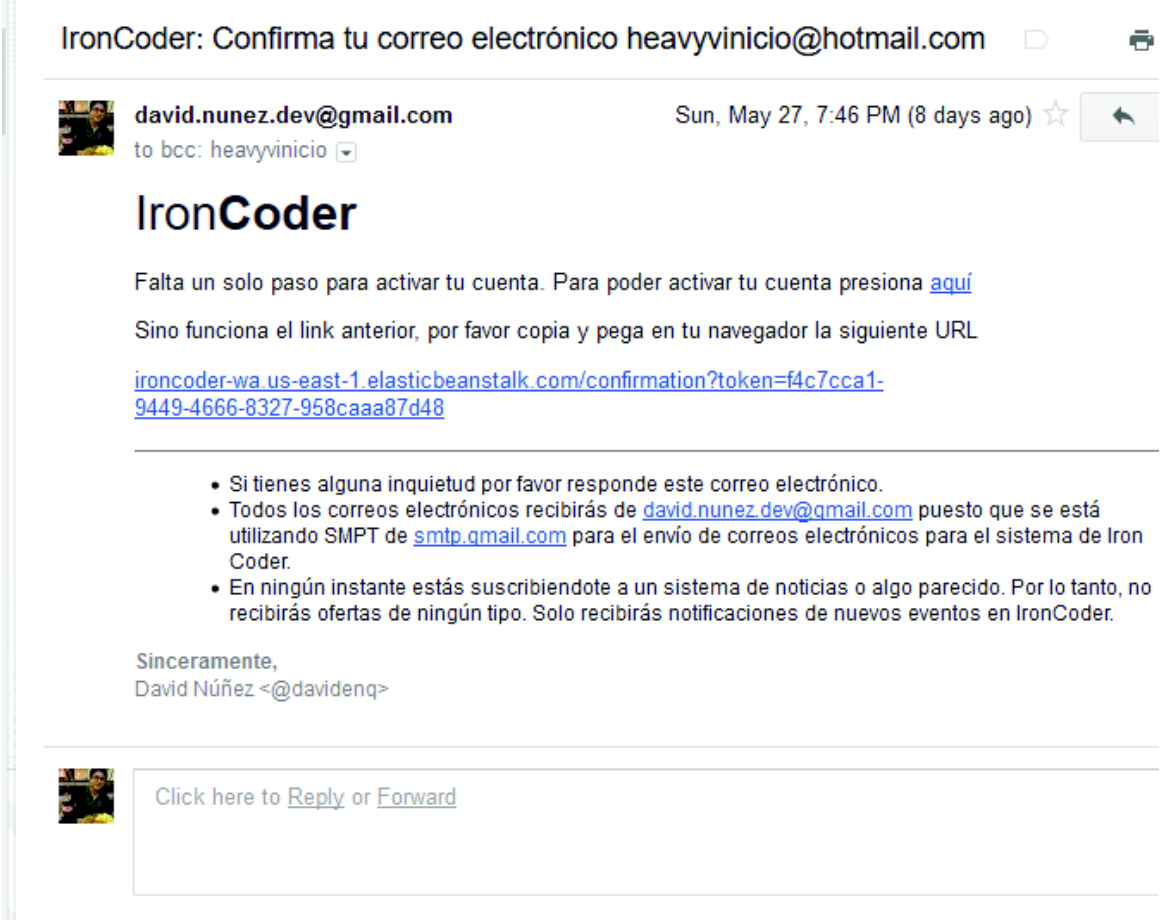


Figura 3.18. Lista de correos enviados a los participantes y contenido

3.1.1 Carpetas de espacio de trabajo creada

Como se observa en la Figura 3.19, el espacio de trabajo `ironcoder` dentro de la carpeta `opt` ha sido creado para las pruebas (`tests`), para el área de competidores (`workspaces`), las dependencias (`vendor`) y los `scripts` para compilar, evaluar y ejecutar el código.

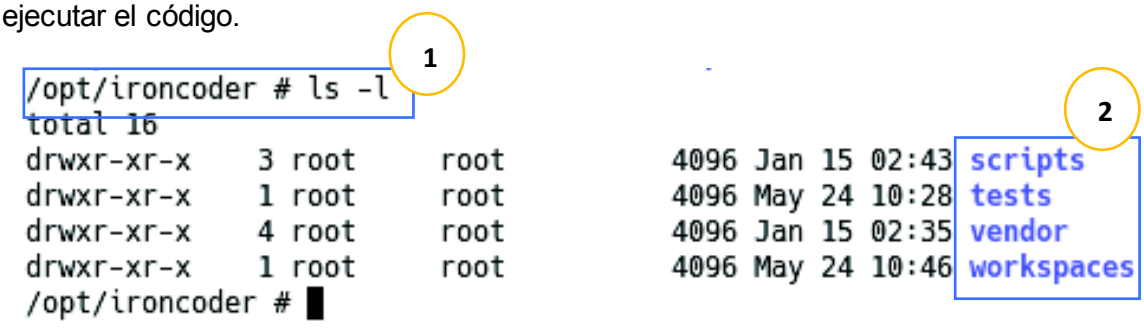


Figura 3.19. Espacio de trabajo de IronCoder creado por el servicio web

3.1.2 Segmentos de la URL de acceso

A continuación, se presentan la descripción del segmento de la URL del servicio web el cual permitirá realizar pruebas sobre dicho servicio para crear espacios de trabajo, crear archivos de código para Java y CSharp y, crear archivos depruebas unitarias.

El segmento de código tiene la siguiente estructura:

`/code/:action/:type` donde:

- `/code:` corresponde a la sección para realizar acciones sobre el código.
- `:action:` está estrechamente ligado al segmento de la URL `:type` y corresponde a la sección para ejecutar tareas ya sea para crear los archivos que contienen las pruebas unitarias o para evaluar el código enviado por los participantes.
- `:type:` está estrechamente ligado al segmento de la URL `:action` y corresponde a la sección para definir la tarea crear la prueba unitaria o de evaluar el código enviado por el participante.

Por ejemplo:

1. `/code/crear/test`
2. `/code/evaluar/challengin-problem`

El primero segmento de URL, permite crear los archivos de prueba, mientras que el segundo, permite evaluar un código que ha sido enviado.

Es importante mencionar que en la segunda opción hay una tarea en *background* que se lleva a cabo y que permite crear el recurso necesario para guardar el código fuente que se evaluará.

Las pruebas realizadas fueron crear espacios de trabajo y crear códigos de prueba. En el Anexo II, se encuentra el conjunto de datos que fue utilizados para cada prueba.

En la Figura 3.20 se puede identificar que en el espacio de trabajo de las pruebas se han creado tres carpetas que contienen las clases de prueba (1).

En (2) se puede identificar que se han creado dos archivos con extensión `.cs` y `.java`. Estos archivos contienen los tests de pruebas para C# y Java, respectivamente.

En (3) se puede observar el código que fue agregado al archivo con extensión `.cs`.

Este código corresponde a la prueba unitaria.

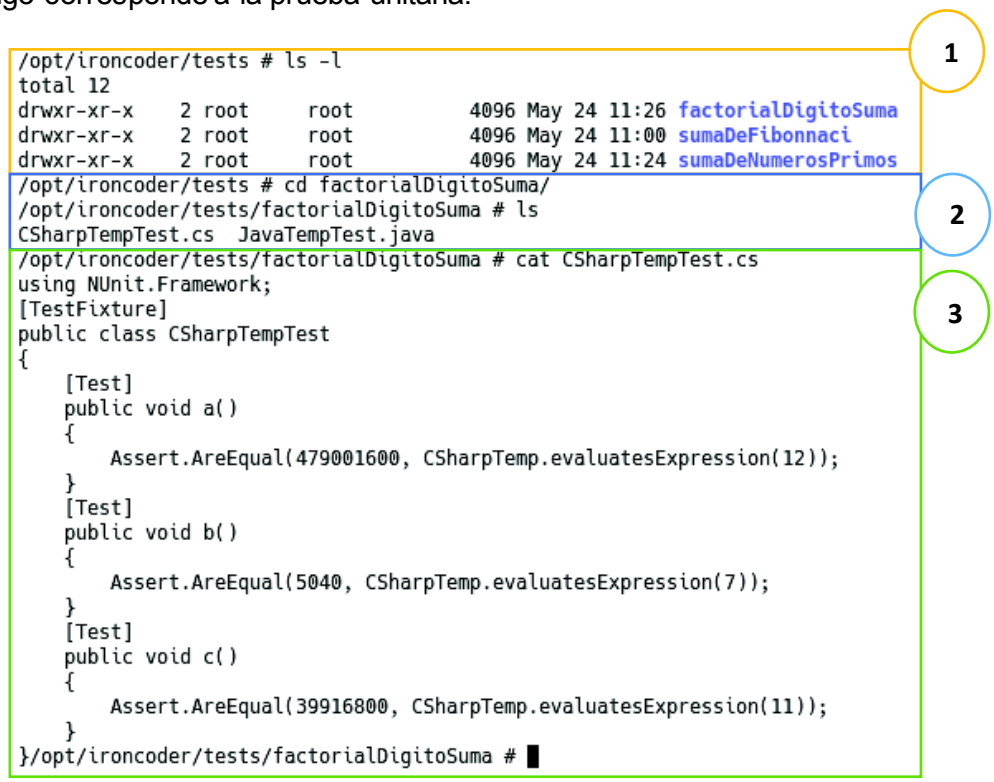


Figura 3.20. Pruebas creadas por el servicio web en la carpeta *tests*

3.3 Torneo de programación

a. Algunas consideraciones que se tuvieron en cuenta

Se obtuvieron un total de 11 participantes.

Ninguno de ellos formó un equipo. fueron competidores únicos.

Las reglas que se dispusieron para el torneo son:

1. Resolver 3 ejercicios, de distintos niveles de dificultad
2. El tiempo de duración del torneo fue programado para 1 hora
3. El límite de participantes por cada equipo fue 2 usuarios por equipo

b. Encuesta realizada

Para llevar a cabo la encuesta, se utilizó el servicio de Google Forms, para crear una encuesta rápida. Las preguntas que fueron incluidas en la encuesta, se encuentran en el Anexo III. Se han considerado una serie de preguntas que están enfocadas en conocer que es lo que piensan los usuarios acerca de los torneos de programación. Adicionalmente, se ha considerado también preguntas acerca del uso de la plataforma.

c. Resultados de la encuesta

La Figura 3.21 muestra el resultado de la primera pregunta. Ésta se realizó con el propósito de obtener una mejor percepción de cuál es la actividad de las personas que decidieron formar parte de las pruebas de la plataforma a través de un torneo de programación.

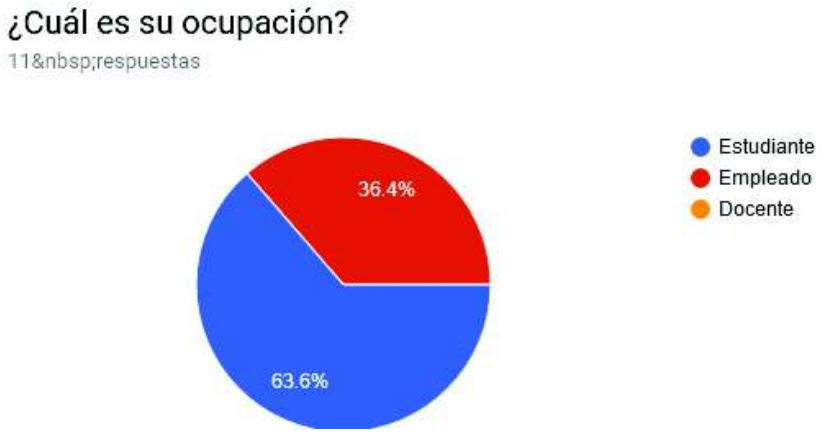


Figura 3.21. Encuesta – pregunta 1

Los estudiantes tienen un mayor porcentaje de participación mientras que las personas que trabajan fueron relativamente menos, considerando que su tiempo es más limitado para este tipo de eventos.

Conocer el nivel que tienen los participantes en cuanto a programación ayuda a determinar que en un próximo evento se considere segmentar de mejor manera los problemas propuestos y el torneo en sí. Los resultados obtenidos se pueden observar en la Figura 3.22.

Figura 3.22. Encuesta – pregunta 2

Hay un gran porcentaje que tienen menos de un año de experiencia resolviendo problemas de programación. Sumado con los resultados de aquellos que recién empiezan a programar o tienen pocos meses haciéndolo, el valor total resulta que es relativamente alto. Es decir, que realmente se debería considerar segmentar muy bien los problemas propuestos.

El resultado de la pregunta 3 se puede observar en la Figura 3.23. El propósito de esta pregunta fue obtener una mejor percepción de cuál es el lenguaje que más se utiliza. En primera posición está Java con un 54.5% mientras que, en la segunda, se encuentra C# con un 36.4%.

Figura 3.23. Encuesta – pregunta 3

El resultado de la pregunta 4 se puede observar en la Figura 3.24. Su propósito es identificar cual es el nivel de aquellos que deciden participar de tal manera que se puedan elegir ejercicios, acorde a los niveles de conocimiento que tienen los participantes.

Figura 3.24. Encuesta – pregunta 4

Las preguntas 5, 6, 7 y 8, con las respuestas que se pueden observar en las Figura 3.25, Figura **3.26**, Figura **3.27** y Figura **3.28**, respectivamente, se realizaron con el propósito de conocer más acerca de la opinión de los tornes de programación. Las respuestas han sido bastante variadas y debatibles.

En general, se puede decir que la mayoría de competidores consideran que los torneos pueden ser un buen mecanismo en el proceso de aprendizaje de programación.

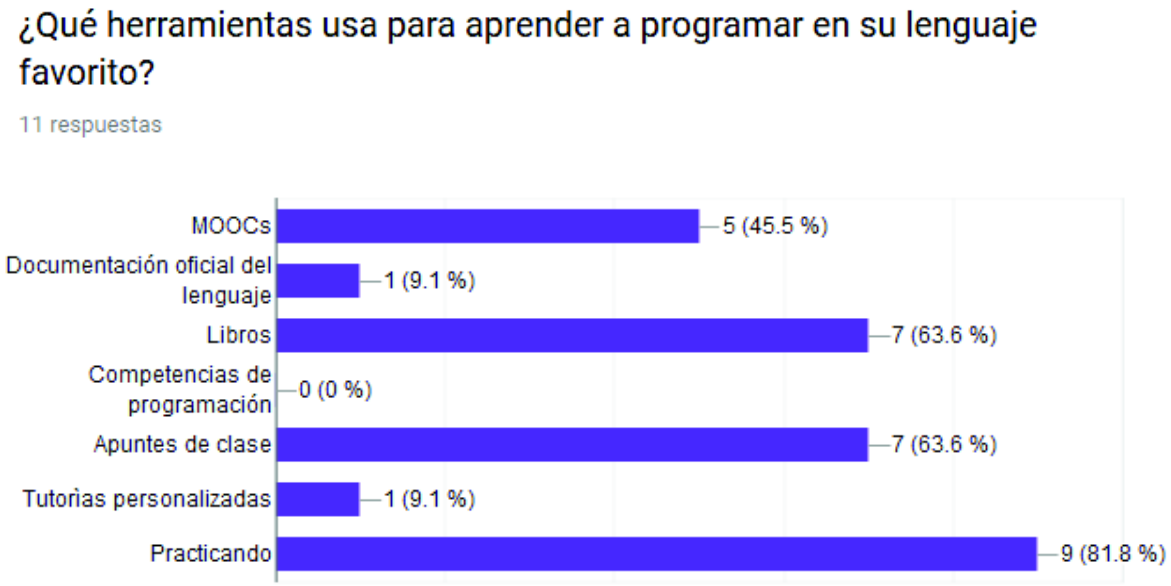


Figura 3.25. Encuesta – pregunta 5

La pregunta 6, fue realizada con el propósito de identificar si con frecuencia se realizan este tipo de eventos que permiten a los programadores ganar habilidades en la resolución de algoritmos. Como se puede observar en la Figura 3.26, el resultado ha sido realmente bajo. Esto se debe a que son pocas las instituciones educativas o privadas, que realizan este tipo de eventos.

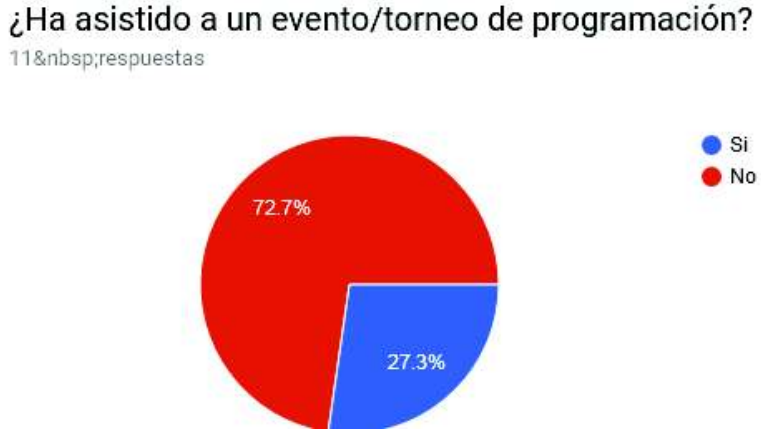


Figura 3.26. Encuesta – pregunta 6

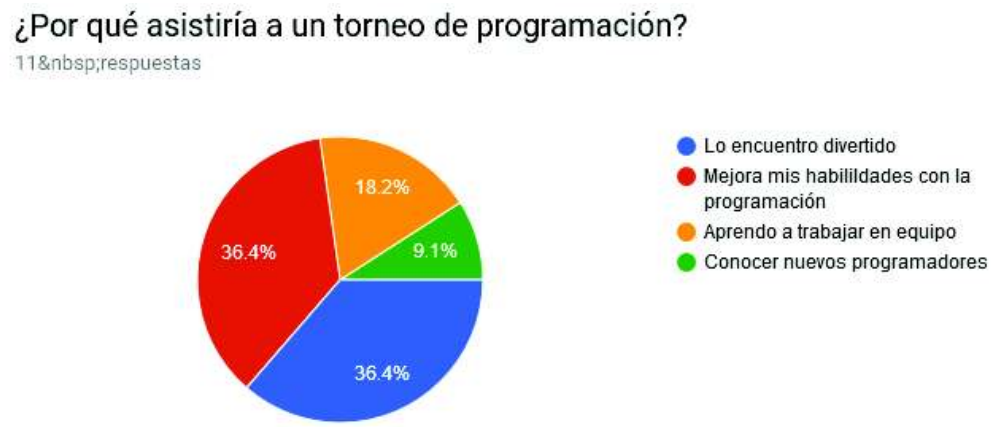


Figura 3.27. Encuesta – pregunta 7

En la pregunta 8, las respuestas han sido bastante variadas y realmente dan un enfoque mucho más especifico acerca de los torneos de programación. Hay varias personas que consideran que los torneos son realmente importantes. Sin embargo, en el ámbito laboral, la resolución de algoritmos, como se menciona en una respuesta, no se ejecuta con mucha frecuencia.

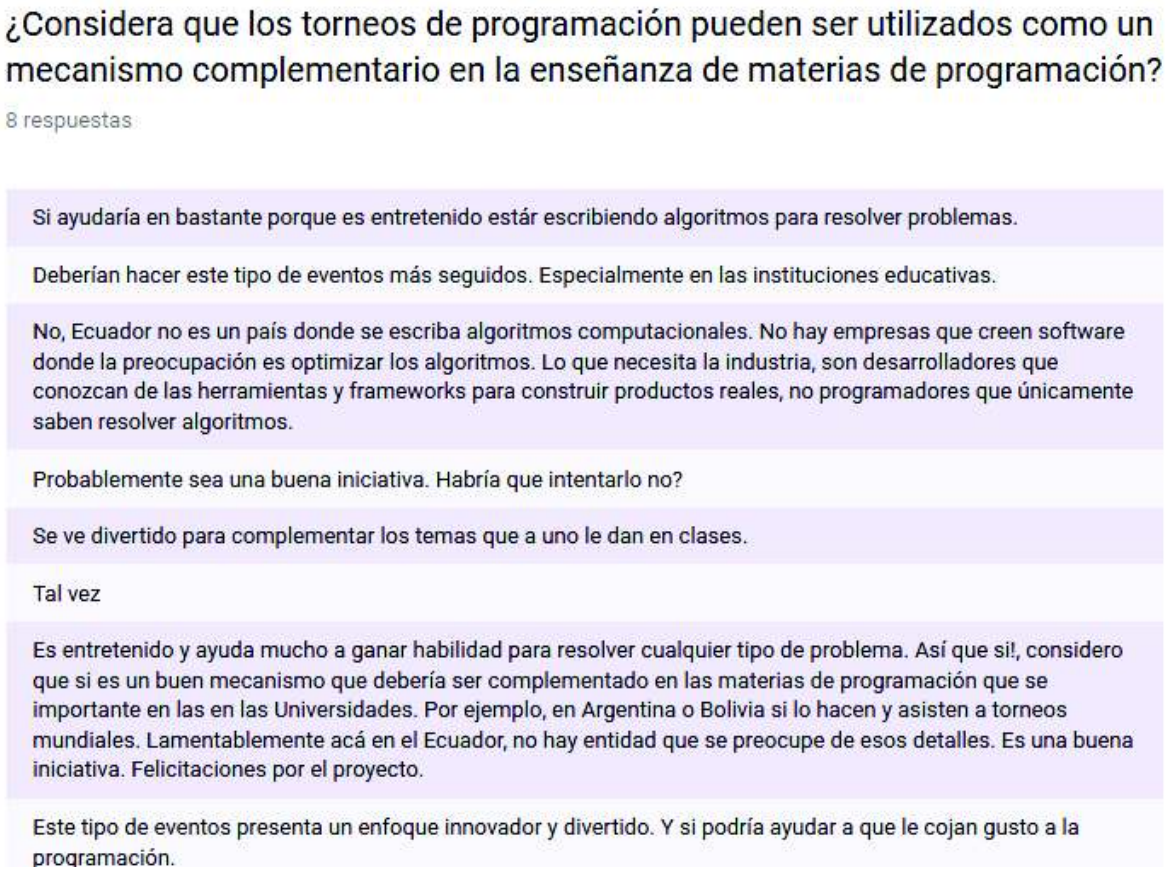


Figura 3.28. Encuesta – pregunta 8

Figura 3.29. Encuesta – pregunta 9

Finalmente, las preguntas 10, 11 y 12 permiten obtener una evaluación final del prototipo puesto a producción para que otros usuarios puedan hacer uso de la misma.

En la Figura 3.30 se muestra el nivel de satisfacción del uso de la plataforma. Se muestra un promedio de un 54.4% quienes seleccionaron el mayor puntaje de valoración. Teniendo como mínimo un puntaje de 7 que representa el 9.1% de los encuestados. El resultado final demuestra que la mayoría de participantes estuvo contento con el uso de la plataforma web.

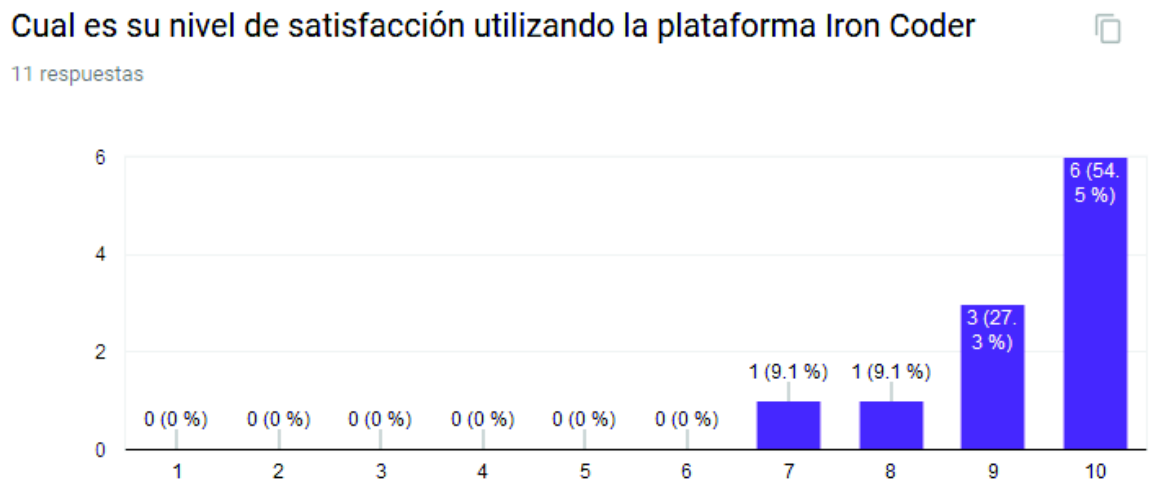


Figura 3.30. Encuesta – pregunta 10

En la Figura 3.31 muestra el resultado del uso de la plataforma. Como se puede observar los valores de satisfacción e insatisfacción están casi a un promedio de un 50%. La experiencia de usabilidad de la misma ha sido relativamente baja y la recomendación dada por un competidor ha sido proveer mayor información.

Esto refleja que en cuanto a funcionalidad y los requerimientos del sistema no tienen nada que ver en el resultado (resultado de la encuesta de problemas de funcionalidad en la Figura 3.32), sino que más bien considerar la experiencia de usuario en el instante de desarrollar un producto es de vital importancia.

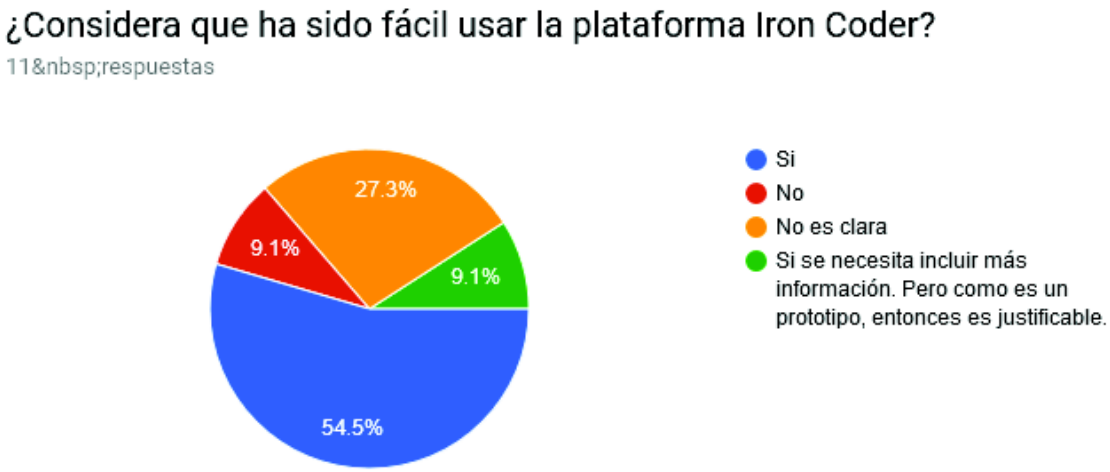


Figura 3.31. Encuesta – pregunta 11

Figura 3.32. Encuesta – pregunta 12

3.4 Discusión

En la siguiente sección se discuten detalles del prototipo, particularmente de las líneas de código que han sido requeridas para poder implementar el prototipo propuesto. También se discuten detalles acerca de algunas incidencias que han sido las más importantes y han tenido un grado de dificultad alto para poder resolverlas. Finalmente se detallan algunos aspectos del uso de recursos y gastos generados utilizando los servicios de AWS.

En el Anexo II se agregan enlaces y una breve descripción acerca de las incidencias que se han suscitado en todo el proceso de implementación del prototipo pero que, sin

embargo, no han tenido un impacto mayor como las incidencias que más adelante se discuten.

a. Líneas de código del proyecto

Se ha realizado un análisis de cuantas líneas de código se han escrito para poder implementar el prototipo propuesto. Para ello, se ha utilizado la herramienta `LOC` la cual se instala con el comando `$sudo apt-get install loc`. El resultado de lectura para el presente proyecto se muestra en la Figura 3.33.

```
davidenq@Home:~/Code/personal$ cloc thesis-loc/
     261 text files.
     253 unique files.
      59 files ignored.

github.com/AlDanial/cloc v 1.70  T=0.93 s (225.5 files/s, 17760.3 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
JavaScript                      37            157            158           4216
Go                              80            824            369           4086
HTML                            35            131             41           2597
CSS                             31            354             24           2474
SQL                              2             55             63            404
JSON                            16              0              0            354
Bourne Shell                     9             13             25            195
-------------------------------------------------------------------------------
SUM:                           210           1534            680          14326
-------------------------------------------------------------------------------
```

Figura 3.33. Líneas de Código del proyecto

Como se puede observar, se han escrito un total de 14.326 líneas de código en los componentes y *scripts* que han sido utilizados tanto en el desarrollo del prototipo, así como en el proceso de despliegue de la aplicación.

En el total de estas líneas de código no se considera el código de terceros ubicado en librerías.

Javascript es el lenguaje en el que más líneas de código se ha escrito. Esto se debe a que cada uno de los componentes web que fueron creados, tienen ya integrado la lógica de negocio que le corresponde a cada componente.

Por ejemplo, el componente `tournament-workspace`, es un componente independiente que solo necesita de los datos enviados por el servidor, para poder mostrar en su interfaz gráfica. Así mismo sucede con los otros componentes web.

El lenguaje de programación Go, también tiene una gran cantidad de líneas de código debido a que no se ha utilizado algún *framework* MVC en particular, sino que se han

construido todas las funcionalidades del sistema, únicamente con las librerías estándar de Golang.

b. Incidencias creación imagen Docker

La construcción de la imagen Docker, a través del archivo `Dockerfile` para el servicio web, presentó una gran dificultad debido a que se seleccionó el sistema operativo Alpine como sistema base para construir la imagen.

Este incidente tuvo que ver con el proceso de instalación del compilador e intérprete Mono para C#. No existe un repositorio estándar definido por el mismo proyecto Mono para poder instalar este *framework* sobre Alpine.

Incluso, no se podía instalar los binarios de Mono a través de la compilación del código fuente base. Esto fue debido a que Mono utiliza la librería de compilación `glib` que es utilizado en la mayoría de distribuciones Linux para compilar código fuente base.

En cambio, Alpine utiliza otra librería de compilación llamada `muslibc`. Por lo tanto, era imposible que Mono sea instalado sobre Alpine. Se consultaron varios foros y los repositorios oficiales de Mono y Alpine para encontrar una solución al problema suscitado.

En (1) de la lista de incidentes del Anexo II se encuentra el enlace para solucionar el problema.

c. Incidencia en el proceso de despliegue

Uno de los incidentes que condujo a un minucioso análisis de si el archivo `Dockerfile` para la aplicación web estaba bien construida o no surgió del hecho de que en el instante de desplegar la aplicación web en el servicio de EBS de AWS, se generaba constantemente una advertencia que impedía que la aplicación pueda ser ejecutada (Figura 3.34).

Después de una exhaustiva investigación, pruebas a nivel local, y refactorización de código, el error se pudo encontrar en el valor de la variable de entorno que era pasado de la instancia de EBS a Docker. Por defecto, en el archivo `Dockerfile`, se configura para que escuche en un puerto en una lista de puertos en particular.

En primera instancia, se exponían dos puertos para que la aplicación pueda escuchar; el puerto 3000 y el puerto 3001. Y en la variable de entorno se configuró el puerto 3001. Sin embargo, basado en la documentación de AWS, EBS, solo escuchará por el primer

puerto que ha sido expuesto en el archivo `Dockerfile`. EBS, no verifica ningún otro puerto expuesto en dicho archivo y la aplicación no podía ser ejecuta, pues prácticamente no había la variable de entorno que expone el puerto por donde debería escuchar.

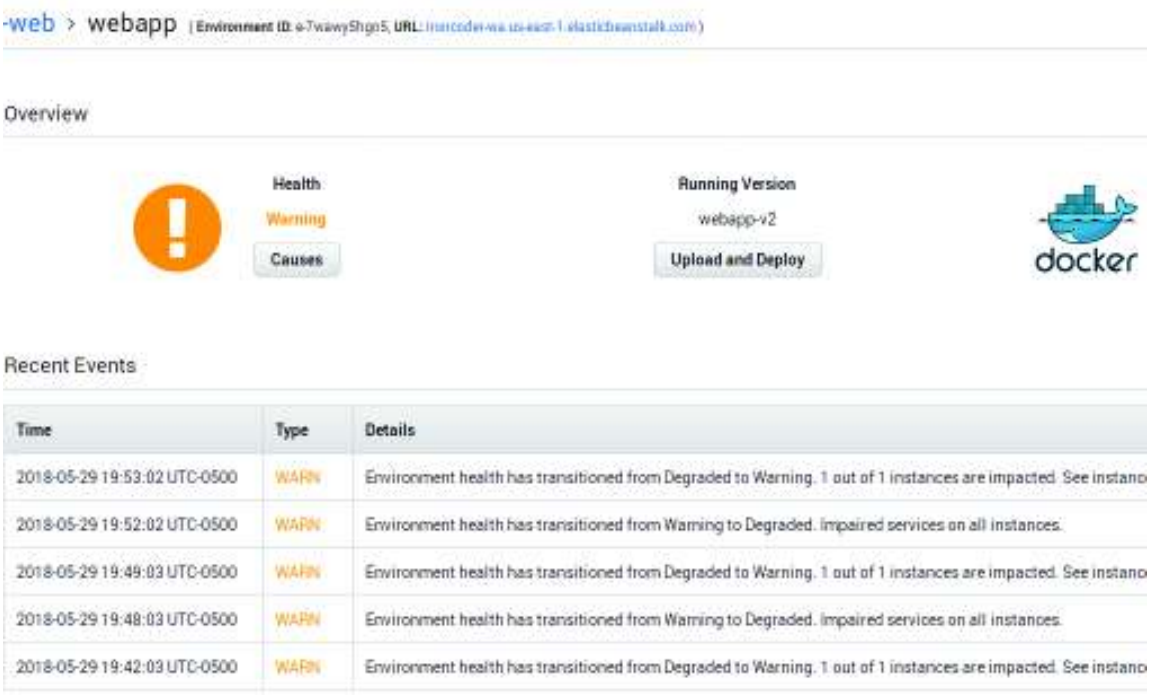


Figura 3.34. *Warning* en el servicio EBS para la aplicación web

d. Niveles de uso de los servicios desplegados en AWS

En la siguiente sección, se desea mostrar algunos detalles con respecto al despliegue de los servicios RDS y EBS en Amazon Web Service. Esta información corresponde a los usos de los recursos desplegados, así como los gastos económicos que han generado.

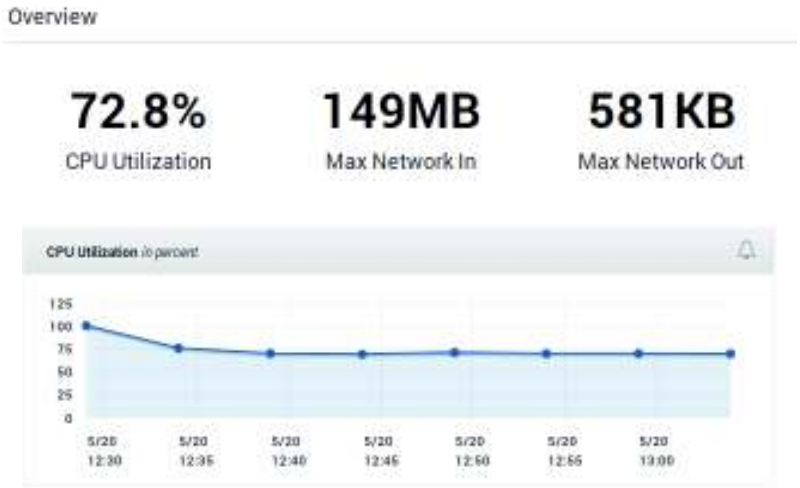


Figura 3.35 Uso de recursos en el despliegue del servicio web.

Como se puede ver en la Figura 3.35, se ha generado un consumo de ancho de banda de 146MB para el servicio web. Este alto consumo de ancho de banda en el proceso de despliegue del servicio web, se debe a que dicho servicio debe instalar las herramientas de Java, JDK y C# con el *runtime* Mono y dichas herramientas tienen un tamaño realmente considerable.

En cambio, como se puede visualizar en la Figura 3.36, el consumo generado en el ancho de banda para el despliegue de la aplicación web, es relativamente bajo, puesto que no se requieren dependencias de gran tamaño.

Figura 3.36. Uso de recursos en el despliegue de la aplicación web.

Figura 3.37. Estadísticas de uso del servicio RDS

En cambio, en el servicio de la base de datos desplegado, el uso se ve reflejado en los datos estadísticos de las operaciones que va ejecutando en el proceso de despliegue y ejecución del servicio (Figura 3.37). El mayor consumo se ha dado aproximadamente el 31 de mayo, teniendo un máximo consumo de memoria de alrededor de 600 MB. Pero este valor no supera el límite permitido para seguir utilizado el servicio de forma gratuita.

e. Costos generados por el despliegue de tres servicios en AWS

Los costos económicos que se han generado en todo el proceso de despliegue y pruebas de la aplicación se pueden ver reflejado en la Figura 3.38. El mes de mayo, es el mes que más costos se ha generado; alrededor de unos $19 han sido facturados por hacer uso de los tres servicios. El valor de consumo diario se puede visualizar en la Figura 3.39. El promedio diario es alrededor $2.40.

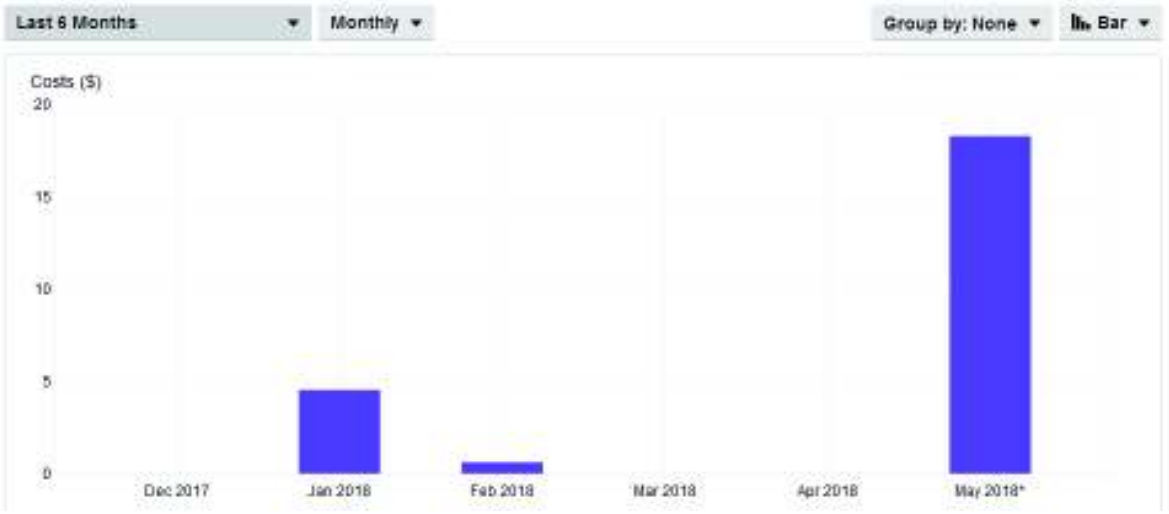


Figura 3.38. Costo económico mensual generados por uso de AWS

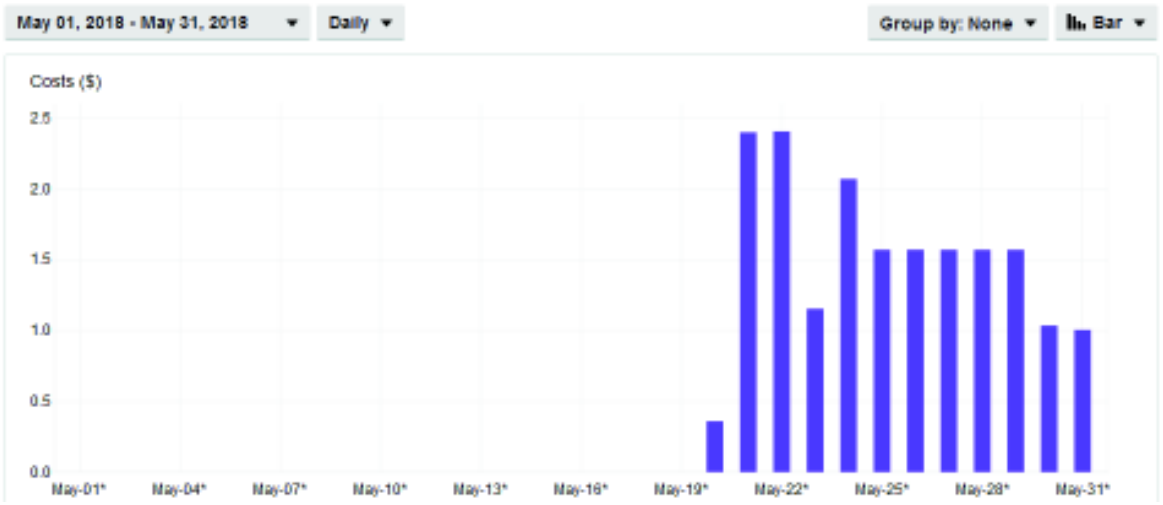


Figura 3.39. Costos económicos diarios por uso de AWS

4 CONCLUSIONES Y RECOMENDACIONES

4.1 Conclusiones

- La metodología de desarrollo ágil Kanban resultó ser bastante fácil de gestionar y ayudó en gran medida a mantener un enfoque de todas las actividades que se llevaron a cabo para implementar el prototipo propuesto. Esta metodología, como se pudo comprobar en el proceso de diseño, análisis e implementación, no requirió gestionar complejas tarjetas o aprender conceptos base para poder utilizarla en el presente proyecto.

- Las pruebas unitarias, consideradas como el mecanismo de evaluación de los algoritmos que son enviados por los competidores en un torneo de programación, han sido beneficiosas puesto que, usando librerías enfocadas en ejecutar pruebas unitarias, la evaluación de código resulta ser mucho más sencilla dado que se abstrae toda la complejidad de crear métodos y clases que permitan definir casos de prueba sobre algún algoritmo y que estos, devuelvan información relevante que muestre el resultado de dicha evaluación.

- Utilizar *scripts* que contienen variables temporales de las rutas, tanto para las clases de prueba, como para las librerías y las clases que son creadas con los algoritmos enviados por los competidores, han sido clave importante para ejecutar tareas de compilación y ejecución de clases pues, dichas variables han ayudado a evitar la complejidad de manejar espacios de nombre para C# y paquetes para Java en el proceso de evaluación del código para un grupo de usuarios.

- El uso de RESTful es efectivo y amigable pues toda la información que es intercambiada entre el servicio web y la aplicación web es en formato JSON el cual no contiene complejidad en su estructura para contener información.

- Presenta un gran beneficio utilizar tecnología de contenedores Docker para empaquetar aplicaciones o servicios puesto que solo se requiere de un archivo, con nombre `Dockerfile`, para que la aplicación o el servicio pueda ser desplegado rápidamente utilizando cualquier servicio en la nube que soporte contenedores Docker.

- Como se ha mencionado en párrafos a lo largo de este documento, contar con un sistema que esté enfocado en los torneos de programación aporta en gran medida

en la preparación de los estudiantes en ganar habilidad en la resolución de algoritmos. Sin embargo, sin la organización donde se promuevan torneos, esta herramienta probablemente solo quede como una referencia de lo cuan complicado puede ser implementar sistemas de este tipo. Además, el hecho de que los problemas que se deben proponer en un torno, deben ser problemas con datos preconstruidos, y ser posteriormente categorizados en un nivel de dificultad, basado en la resolución previa de dicho problema, considerando tiempos y nivel de conocimiento de un lenguaje, demanda tiempo y que probablemente, esto sea una limitación para que se realicen torneos.

- El realizar un análisis acerca del patrón MVC y la modularización, ha permitido entender de mejor manera como es la separación de responsabilidades de cada una de las capas que conforma dicho patrón y como esto ayuda a modularizar el código cuando no se utiliza un *framework* para el desarrollo de aplicaciones o servicios web. Esto se ha podido identificar tanto en el análisis de dicho patrón, así como en el proceso de modularización del prototipo.

- Los *scripts* que permiten llevar a cabo las tareas de compilación, ejecución y evaluación del código utilizando todas las dependencias requeridas para dichas acciones, son la parte más importante de los servicios web.

- Gracias al uso de servicios web, la complejidad de implementar un sistema que maneje toda la lógica de evaluación, creación de espacios de trabajo, creación de archivos de prueba, etc., quedó centralizada únicamente el servicio creado y el cual permitió que fuese desarrollado de forma totalmente independiente a la aplicación web. Así mismo, el uso de componentes permitió que las interfaces creadas que contienen varias funcionalidades, tuvieran un desarrollo totalmente independiente de la aplicación. La integración de dichos componentes quedó limitada únicamente a agregar como recursos estáticos de la página donde se quería disponer de dicho componente, por ejemplo, el componente web que presenta la interfaz de los torneos de programación.

- Para implementar el prototipo propuesto, se requirió previamente realizar un exhaustivo estudio de la tecnología Docker, el uso de los servicios de Amazon, librerías y herramientas, además adquirir un conocimiento lo suficientemente amplio de los lenguajes de programación Go y Javascript. Pues todos estos temas no son impartidos en las materias de la Carrera de Ingeniería en Electrónica y Redes de Información. Esto se evidenció a lo largo de este documento.

4.2 Recomendaciones

- Para trabajos de desarrollo de software donde está involucrado un único desarrollador, se recomienda usar la metodología ágil Kanban. Esto es gracias a la facilidad que presenta para gestionar las actividades de un proyecto.
- Mantener una bitácora de información de las actividades que se van realizando y las incidencias que se van presentando en cada tarea llevada a cabo. Esto con el propósito de ser utilizado más delante de tal manera que se pueda documentar la información que sea relevante para un posterior análisis.
- En lo que respecta a desarrollar aplicaciones como la del presente proyecto, que involucran varias áreas de trabajo (*backend*, *frontend* y despliegue), se debería considerar formar un equipo de trabajo antes de empezar el desarrollo. Esto permitiría distribuir las actividades de cada área, a cada integrante del equipo y cada uno de ellos estaría enfocado en una única área, reduciendo así la carga de trabajo que implicaría si solo una única persona se encarga de todas las áreas.
- Realizar un previo estudio de las herramientas y tecnologías que se desea utilizar para implementar un prototipo; verificando si hay una buena documentación, si hay una comunidad activa y si su uso contribuirá de alguna manera a un desarrollo rápido de tal manera que los tiempos sean mínimos. Esto con el propósito de evitar que, cuando se presenten incidencias en el desarrollo, no tome mucho tiempo resolverlas las incidencias que se presenten considerando que, para encontrar una solución, la documentación ayude en gran medida y también exista una comunidad activa que esté presta para ayudar a solucionar el problema que se presente.
- Utilizar lenguajes de programación fuertemente tipados si es que se desea extender la funcionalidad de este prototipo a nuevos lenguajes de programación. Esto se debe a que los *scripts* de ejecución, el componente para crear el problema con casos de prueba, y las clases que crean nuevas clases con el código de los participantes, son diseñados para manejar lenguajes de programación fuertemente tipados.
- Si se requiere extender la funcionalidad del prototipo para que soporte tipos de datos complejos, se recomienda realizar un exhaustivo estudio de las librerías que son utilizadas para prueba unitarias y como estas, utilizan métodos avanzados para lleva a cabo pruebas sobre estructuras de datos complejas.

- Tecnologías de contenedores puedan ser consideradas para ser impartidas en la Facultad en lugar de máquinas virtuales. Esto, considerando que, en la actualidad, Docker es comúnmente utilizado para realizar tareas de despliegue de servicios y aplicaciones en la nube abstrayendo evitando problemas de compatibilidad de los entornos de desarrollo y producción. Así mismo, el uso de *scripts* vía *shell* debería tener una consideración importante dentro de la Carrea de Electrónica y Redes de Información pues, como tal, un ingeniero en Redes debe aprender a gestionar y automatizar tareas dentro servidores, servicios, sistemas operativos utilizando la línea de comandos y *scripts.*

5 REFERENCIAS BIBLIOGRÁFICAS

[1] S. Halim y F. Halim, “Competitive Programming 3”, *Lulu Press Inc.*, p. 448, 2013.

[2] M. A. Skiena, Steven S., Revilla, *Programming Challenges*, Springer. New York: Springer, 2003.

[3] ITMO University, “5 Career Benefits of Competitive Programming | edX Blog”, 2017. [En línea]. Disponible en: https://blog.edx.org/5-career-benefits-of-competitive-programming. [Consultado: 28-ene-2018].

[4] Duncan Smith, “12 Reasons to Study Competitive Programming - Red-Green-Code”, 2015. [En línea]. Disponible en: https://www.redgreencode.com/12-reasons-to-study-competitive-programming/.[Consultado: 27-ene-2018].

[5] M. Guerrero, “Integración de un conjunto de herramientas telemáticas en una solución de apoyo al proceso de aprendizaje de la asignatura de programación en la FIEE”, vol. 1, p. 176, 2015.

[6] D. Gries *et al.*, “Distributed Applications”, *New Comput. Paradig. Comput. Music*, vol. 54, núm. 3, pp. 1–159, 2010.

[7] A. Rodriguez, “RESTful Web services: The basics”, New York, *Press Inc,* 2015.

[8] BBVAOPEN4U, “‘APIs for dummies’: cinco maneras de descubrir qué es una API”, 2018. [En línea]. Disponible en: https://bbvaopen4u.com/es/actualidad/apis-dummies-cinco-maneras-de-descubrir-que-es-una-api. [Consultado: 02-feb-2018].

[9] L. Shklar y R. Rosen, *Web Application Architecture: Principles, Protocols and Practices*. Canada: Distripack, 2003.

[10] Y. Shoitsu, “An Introduction to Web Components”. [En línea]. Disponible en: https://www.upwork.com/hiring/development/web-components/. [Consultado: 22-ene-2018].

[11] N. C. Zakas Jeremy McPeak Joe Fawcett, N. C. Professional Ajax, N. C. Zakas, J. McPeak, y J. Fawcett, “Professional Ajax Library of Congress Cataloging-in-Publication Data”.

[12] R. T. Fielding, “Architectural Styles and the Design of Network-based Software Architectures” https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm,

[Consultado: 22-ene-2018].

[13] C. Jendrock, Eric, Cervera-Navarro, Ricardo, Evans, Ian, Gollapudi, Devika, Haase, Kim, Markito, William, Srivathsa, “The Java EE 6 Tutorial”, 2013. [En línea]. Disponible en: https://docs.oracle.com/javaee/6/tutorial/doc/index.html. [Consultado: 12-feb-2017].

[14] M. Pilar del Saz Portillo, “Tutorial patrón MVC”. [En línea]. Disponible en: http://www.codigonexo.com/wp-content/uploads/2014/06/Curso-completo-MVC.pdf. [Consultado: 26-feb-2018].

[15] D. Núñez, “Entendiendo M de MVC y sus problemas – @davidenq – Medium”, 2016. [En línea]. Disponible en: https://medium.com/@davidenq/entendiendo-m-de-mvc-y-sus-problemas-ebc0cbf518ec. [Consultado: 22-abr-2018].

[16] K. B. Clark *et al.*, “Modularity in the Design of Complex Engineering Systems”, Baldwin: Academic Press, 2004.

[17] A. Böckmann, M. Lanza, y M. Lungu, “MARS -Modular Architecture Recommendation System Analysis of System Decompositions through Coupling and Cohesion metrics”, McGraw-Hill, 2010.

[18] M. Stauffer, “Laravel 5.0 - Directory structure and namespace | MattStauffer.com”, 2014. [En línea]. Disponible en: https://mattstauffer.com/blog/laravel-5.0-directory-structure-and-namespace/. [Consultado: 23-feb-2018].

[19] P. Padmanabhan, Senthil, Steel-Idem, “Don’t Build Pages, Build Modules”, 2014. [En línea]. Disponible en: https://www.ebayinc.com/stories/blogs/tech/dont-build-pages-build-modules/. [Consultado: 13-ene-2018].

[20] R. Vukasinovic, “Building modular web interfaces – Hacker Noon”. [En línea]. Disponible en: https://hackernoon.com/building-modular-interfaces-a4e4076b4307. [Consultado: 13-feb-2018].

[21] P. SteelIdem, “Applying Modularity to the Front-End”, 2014. [En línea]. Disponible en: https://es.slideshare.net/PatrickSteeleIdem/applying-modularity-to-the-frontend-40924987. [Consultado: 02-feb-2018].

[22] Docker Inc., “Docker For Windows | Docker”, 2018. [En línea]. Disponible en: https://www.docker.com/docker-windows. [Consultado: 21-abr-2018].

[23] A. N. Excerpt, F. Docker, I. N. Action, y B. Y. Jeff, “USING DOCKER TO BUILD A

WEBSITE MONITOR, Ed. San Diego: Academic Press, 1995, pp. 133-180

[24] D. Vohra, Pro Docker, 2nd. ed. New York: McGraw-Hill, 2004.

[25] N. Agarwal, “Lifecycle of Docker Container – Nitin Agarwal – Medium”. [En línea]. Disponible en: https://medium.com/@nagarwal/lifecycle-of-docker-container-d2da9f85959. [Consultado: 12-mar-2018].

[26] E. Ubl, Malte, Kitamura, “Introducing WebSocket: Bringing Sockets to the Web - HTML5 Rocks”, 2010. [En línea]. Disponible en: https://www.html5rocks.com/es/tutorials/websockets/basics/. [Consultado: 19-mar-2018].

[27] J. Schlichter, “Distributed Applications”, *Distrib. Appl. Scr.*, New York: Cambridge University Press, March, 2002.

[28] G. Bigby, “The 6 Types of HTTP Status Codes Explained”, 2018. [En línea]. Disponible en: https://dynomapper.com/blog/254-the-6-types-of-http-status-codes-explained. [Consultado: 01-abr-2018].

[29] C. Reichert, “7 HTTP methods every web developer should know and how to test them : Assertible”. [En línea]. Disponible en: https://assertible.com/blog/7-http-methods-every-web-developer-should-know-and-how-to-test-them. [Consultado: 01-abr-2018].

[30] L. Wagner, Bill, Wenzel, Maira, Levin Isaac, Latham, “Introduction to the C# Language and the .NET Framework | Microsoft Docs”, 2015. [En línea]. Disponible en: https://docs.microsoft.com/en-us/dotnet/csharp/getting-started/introduction-to-the-csharp-language-and-the-net-framework. [Consultado: 04-abr-2018].

[31] C. Jendrock, Eric, Cervera-Navarro, Ricardo, Evans, Ian, Gollapudi, Devika, Haase, Kim, Markito, William, Srivathsa, *The Java EE 6 Tutorial*, 2d Edition., núm. March. Redwood, 2013.

[32] The Linux Information Project, “Shell definition by The Linux Information Project (LINFO)”. [En línea]. Disponible en: http://www.linfo.org/shell.html. [Consultado: 04-abr-2018].

[33] R. Cheplyaka, “shell - Difference between sh and bash - Stack Overflow”, 2011. [En línea]. Disponible en: https://stackoverflow.com/a/5725402/4414954. [Consultado: 05-abr-2018].

[34] R. Pike, “Go at Google: Language Design in the Service of Software Engineering”, 2012. [En línea]. Disponible en: https://talks.golang.org/2012/splash.article. [Consultado: 08-abr-2018].

[35] K. Dudeja, “The beauty of Go – Hacker Noon”, 2017. [En línea]. Disponible en: https://hackernoon.com/the-beauty-of-go-98057e3f0a7d. [Consultado: 12-mar-2018].

[36] C. Doxsey, “Concurrency — An Introduction to Programming in Go | Go Resources”, 2012. [En línea]. Disponible en: https://www.golang-book.com/books/intro/10. [Consultado: 13-abr-2018].

[37] A. Edwards, “Serving Static Sites with Go”, 2017. [En línea]. Disponible en: https://www.alexedwards.net/blog/serving-static-sites-with-go. [Consultado: 16-feb-2018].

[38] S. Manohar, “Understanding Go’s template package – Satish Manohar Talim – Medium”, 2015. [En línea]. Disponible en: https://medium.com/@IndianGuru/understanding-go-s-template-package-c5307758fab0. [Consultado: 08-mar-2018].

[39] Y. Pitsishin, “Example Of Using Templates In Golang”, 2016. [En línea]. Disponible en: http://goinbigdata.com/example-of-using-templates-in-golang/. [Consultado: 08-mar-2018].

[40] A. Kondov, “Entender JS: The Event Loop - Hacker mediodía”, 2017. [En línea]. Disponible en: https://hackernoon.com/understanding-js-the-event-loop-959beae3ac40. [Consultado: 15-abr-2018].

[41] P. Roberts, “Philip Roberts: What the heck is the event loop anyway? | JSConf EU 2014 - YouTube”, 2014.

[42] Alex Kyriakidis, Kostas Maniatis, Evan You, A. Kyriakidis, K. Maniatis, y E. You, “The Majesty of Vue . js 2”, Piscataway, NJ, USA: p. 297, 2017.

[43] A. Gore, “4 Things Vue.js Got Right – Vue.js Developers – Medium”, 2016. [En línea]. Disponible en: https://medium.com/js-dojo/4-things-vue-js-got-right-10820cc84004. [Consultado: 20-ene-2018].

[44] Bemenderfer Joshua, “Writing Custom Vue.js Directives ← Alligator.io”, 2016. [En línea]. Disponible en: https://alligator.io/vuejs/custom-directives/. [Consultado: 19-feb-2018].

[45] M. Lahin, “Vee-validate(Intro and Example) – Muhammad Lahin – Medium”, 2017. [En línea]. Disponible en: https://medium.com/@lahin31/vee-validate-intro-and-example-22d8b95e25e1. [Consultado: 24-abr-2018].

[46] T. Petric, “Managing configuration with Viper”, 2017. [En línea]. Disponible en: https://scene-si.org/2017/04/20/managing-configuration-with-viper/. [Consultado: 08-may-2018].

[47] P. Bengtsson, “My favorite Go multiplexer - Peterbe.com”, 2015. [En línea]. Disponible en: https://www.peterbe.com/plog/my-favorite-go-multiplexer. [Consultado: 08-may-2018].

[48] Vogella Gmbh, “Unit Testing with JUnit - Tutorial”, 2017. [En línea]. Disponible en: http://www.vogella.com/tutorials/JUnit/article.html. [Consultado: 29-ene-2018].

[49] Kumar Dhananjay, “Getting started with .NET unit testing using NUnit | Infragistics Blog”, 2015. [En línea]. Disponible en: https://www.infragistics.com/community/blogs/b/dhananjay_kumar/posts/getting-started-with-net-unit-testing-using-nunit. [Consultado: 29-ene-2018].

[50] D. Neiding, “Production, Test & Development Environments | Razorleaf”, 2009. [En línea]. Disponible en: https://www.razorleaf.com/2009/12/prod-test-dev-environments/. [Consultado: 16-abr-2018].

[51] K. Carias, “Simple words for a GitLab Newbie | GitLab”, 2015. [En línea]. Disponible en: https://about.GitLab.com/2015/05/18/simple-words-for-a-GitLab-newbie/. [Consultado: 21-mar-2018].

[52] M. Rouse, “What is Amazon Web Services (AWS)”, 2013. [En línea]. Disponible en: https://searchaws.techtarget.com/definition/Amazon-Web-Services. [Consultado: 24-mar-2018].

[53] B. Holland, “Visual Studio Code Can Do That? — Smashing Magazine”, 2018. [En línea]. Disponible en: https://www.smashingmagazine.com/2018/01/visual-studio-code/. [Consultado: 25-abr-2018].

[54] J. Wallen, “Review: MySQL Workbench - TechRepublic”, 2010. [En línea]. Disponible en: https://www.techrepublic.com/blog/product-spotlight/review-mysql-workbench/. [Consultado: 28-abr-2018].

[55] P. Wanyoike, Michael, Dierx, “A Beginner’s Guide to npm — the Node Package Manager”, 2017. [En línea]. Disponible en: https://www.sitepoint.com/beginners-

guide-node-package-manager/. [Consultado: 29-abr-2018].

[56] A. Gore, “Boilerplates/Templates For VueJS Projects”, 2018. [En línea]. Disponible en: https://vuejsdevelopers.com/2018/04/23/vue-boilerplate-template-scaffold/. [Consultado: 30-abr-2018].

[57] McFarlin Tom, “The Beginner’s Guide to Unit Testing: What Is Unit Testing?”, 2012. [En línea]. Disponible en: https://code.tutsplus.com/articles/the-beginners-guide-to-unit-testing-what-is-unit-testing--wp-25728. [Consultado: 21-feb-2018].

[58] D. Radigan, “Kanban - El orientador ágil”, 2017. [En línea]. Disponible en: https://es.atlassian.com/agile/kanban. [Consultado: 01-may-2018].

[59] M. Weinberger, “CodeFights thinks competitive programming can be more popular than football - Business Insider”, 2015. [En línea]. Disponible en: http://www.businessinsider.com/codefights-thinks-competitive-programming-can-be-a-spectator-sport-2015-9. [Consultado: 01-may-2018].

[60] R. Dua, S. K. Konduri, y V. Kohli, “Learning Docker networking”, Canadá, Apress, 2016.

[61] C. L. Lapuente y M. J. L. Lapuente, “DHTML” https://stackoverflow.com/a/DHTML-98988osioe-DTHML/4414954. [Consultado: 05-abr-2018].

[62] Lengstorf Jason, “JSON: What It Is, How It Works, & How to Use It - Copter Labs”, *Journal*, 2015. [En línea]. Disponible en: https://www.copterlabs.com/json-what-it-is-how-it-works-how-to-use-it/. [Consultado: 06-ene-2018].

[63] H. Schildt, “Java ™: The Complete Reference, Seventh Edition”, San Diego: Academic Press, 2017.

[64] Project Mono, “Documentation | Mono”, *2018*, 2018. [En línea]. Disponible en: https://www.Mono-project.com/docs/. [Consultado: 03-abr-2018].

6 ANEXOS

ANEXO I. *Scripts*

ANEXO II. Lista de incidencias

ANEXO III. Preguntas de la encuesta realizada

ANEXO IV. Manual de uso

ANEXO V. Capturas de pantalla de tablas de la base de datos

ANEXO VI. Código

ANEXO I

Scripts

Compilación y ejecución de pruebas unitarias para C#

```
NAMEGROUP=$1
NAMEPROBLEM=$2
IRONCODER_PATH="/opt/ironcoder"
NUNIT_VERSION="3.9.0"
RUNNER_VERSION="3.7.0"


TESTS="$IRONCODER_PATH/tests/$NAMEPROBLEM"
USERSPACE="$IRONCODER_PATH/workspaces/$NAMEGROUP/$NAMEPROBLEM"
VENDOR="$IRONCODER_PATH/vendor"

#set tools nunit path
NUNIT_PATH="$VENDOR/dotnet"
NUNIT_LIBRARY="$NUNIT_PATH/NUnit.$NUNIT_VERSION/lib"
NUNIT_FRAMEWORK="$NUNIT_LIBRARY/net45/nunit.framework.dll"
NUNIT_CONSOLE="$NUNIT_PATH/NUnit.ConsoleRunner.$RUNNER_VERSION/tools/nunit3-console.exe"

NUNIT_RUNNERS="$NUNIT_PATH/NUnit.ConsoleRunner.$RUNNER_VERSION/tools"

export MONO_PATH=$USERSPACE:$NUNIT_RUNNERS:$NUNIT_LIBRARY/net45
# compile CSharpTemp class as a library
mcs "$USERSPACE/CSharpTemp.cs" -t:library
# compile CSharpTempTest and CSharpTemp classes as an unique library
mcs "$TESTS/CSharpTempTest.cs" -r:"$NUNIT_FRAMEWORK,$USERSPACE/CSharpTemp.dll" -out:"$TESTS/CSharpTempTest.dll" -t:library
# execute test
mono $NUNIT_CONSOLE "$TESTS/CSharpTempTest.dll" -noresult |
grep -e 'Duration*.*seconds$' -e 'Test Count*.*' | sed -e ':a;N;$!ba;s/\n/, /g'
#clear out
find $USERSPACE $TESTS -name "*.dll" -type f -delete
find $USERSPACE $TESTS -name "*.exe" -type f -delete
rm -rf InternalTrace* nunit-agent* TestResult*
```

Código para dar formato a las respuestas enviadas desde el CLI

```
package utils

import (
    "remex"
    "strings"
)

type dataTest map[string]string

/*
type dataTest struct {
    NumberTests string //number
    Passed      string //number
    Failed      string //number
    Duration    string //in seconds
}*/

//FromString ...
func (dt dataTest) FromString(result string, lang string) dataTest {

    var outcome dataTest
```

```
    if lang == "CSharp" || lang == "C#" {
        outcome = dt.searchDataCSharp(result)
    } else {
        outcome = dt.searchDataJava(result)
    }
    return outcome
}

func (dt dataTest) searchDataJava(data string) dataTest {

    /*
        when a code pass all tests, then data is like:
        Time: 0,004, OK (3 tests)
        Time: 0,006, Tests run: 3,  Failures: 3
    */
    temp := strings.Split(data, ",")
    if strings.Contains(data, "OK") {
        regex := regexp.MustCompile("[0-9]+")
        number := regex.FindAllString(temp[1], -1)
        return map[string]string{
            "NumberTests": number[0],
            "Passed":      number[0],
            "Failed":      "0",
            "Duration":    strings.Split(temp[0], ":")[1],
        }

    }
    return map[string]string{
        "NumberTests": strings.Split(temp[1], ":")[1],
        "Passed":      "0",
        "Failed":      strings.Split(temp[2], ":")[1],
        "Duration":    strings.Split(temp[0], ":")[1],
    }
}

func (dt dataTest) searchDataCSharp(data string) dataTest {
    /*
        indepently if a code pass or fail all tests, then data is like:
        Test Count: 3, Passed: 3, Failed: 0, Warnings: 0, Inconclusive: 0, Skipped: 0,
Duration: 0.909 seconds
        arr will be like:
        TestCount:3
        Passed:3
        Failed:0
        Warnings:0
        Inconclusive:0
        Skipped:0
        Duration:0.872seconds
    */
    temp := strings.Split(data, ",")

    return map[string]string{
        "NumberTests": strings.Split(temp[0], ":")[1],
        "Passed":      strings.Split(temp[1], ":")[1],
        "Failed":      strings.Split(temp[2], ":")[1],
        "Duration":    strings.Split(temp[6], ":")[1],
    }
```

ANEXO II

Lista de incidencias

1. **Incidencia 1:** Soporte del *runtime* Mono para Alpine/Linux

   Descripción: En lugar de utilizar el repositorio master se utilizó el repositorio de pruebas que contiene nuevos soportes para alpine. La URL de testing de mono es http://dl-4.alpinelinux.org/alpine/edge/testing la solución planteada se encontró en la URL que a continuación se menciona.

   Solución: https://github.com/gliderlabs/docker-alpine/issues/31#issuecomment-169370032

2. **Incidencia 3:**

   ¿Cómo usar llaves SSH en Docker en el proceso de despliegue en AWS para clonar los repositorios?

   La solución fue no agregar la clave de acceso cuando se crear las llaves SSH. Los enlaces a la documentación, foros y blogs a los que se accedieron se encuentran listados a continuación:

   - https://farazdagi.com/2016/using-ssh-private-keys-securely-when-building-docker-images/
   - https://unix.stackexchange.com/questions/12195/how-to-avoid-being-asked-passphrase-each-time-i-push-to-bitbucket
   - https://forum.GitLab.com/t/ssh-unsupported-option-rsaauthentication/11198/3
   - https://bbs.archlinux.org/viewtopic.php?id=226080

   Solución encontrada en: https://GitLab.com/davidenq/webservices.git

3. **Incidencia 4:** Binarios de Go no funcionan en Alpine/Linux

   Dado que Ubuntu utiliza glibc para compilar mientras que Alpine usa musl

   La solución fue utilizar una imagen pre-construida por la comunidad.

   Solución encontrada en: https://github.com/frol/docker-alpine-glibc

4. **Incidencia 5:** Variables de entorno para Java y C# no son exportadas de forma predeterminada en Alpine.

   Se solucionó esta incidencia agregando enlaces simbólicos a las rutas de los binarios de Java y Mono.

   https://unix.stackexchange.com/questions/176027/ash-profile-configuration-file

   Solución encontrada en: https://stackoverflow.com/questions/35325856/where-to-set-system-default-environment-variables-in-alpine-linux#comment63491121_35357011

ANEXO III

Preguntas de la encuesta realizada

1. ¿Cuál es su ocupación?

    - Estudiante
    - Empleado
    - Docente

2. ¿Cuánto tiempo lleva programando?

    - Recién empiezo
    - Unos pocos meses
    - Menos de un año
    - Entre uno y dos años
    - Entre dos y cinco años
    - Más de cinco años

3. Sobre que lenguaje de programación trabaja actualmente

    - Java
    - C#
    - C++
    - Python
    - Ruby
    - Javascript
    - Haskell
    - Php

4. Su nivel de conocimiento sobre el lenguaje que seleccionó para el torneo es
    - Entre 1 y 10%
    - Entre 10 y 50%
    - Entre 50 y 80%
    - Soy una eminencia
    - No sé cómo evaluar mi conocimiento

5. ¿Qué herramientas usa para aprender a programar en su lenguaje favorito?

    - MOOCs
    - Documentación oficial del lenguaje
    - Libros
    - Competencias de programación
    - Apuntes de clase
    - Tutorías personalizadas
    - Practicando

6. ¿Conoce acerca de eventos de programación?

- Si
- No

7. ¿Ha asistido a un evento/torneo de programación?

   - Si
   - No

8. ¿Por qué asistiría a un torneo de programación?

   - Lo encuentro divertido
   - Mejorar mis habilidades de programación
   - Aprender a trabajar en equipo
   - Conocer nuevos programadores

9. ¿Considera que los torneos de programación pueden ser utilizados como un mecanismo complementario en la enseñanza de materias de programación? ¿Por qué?

10. ¿Considera que ha sido fácil usar la plataforma Iron Coder?

    - Si
    - No

11. ¿Tuvo algún inconveniente utilizando la plataforma Iron Coder?
    - Si
    - No

ANEXO IV

Manual de uso

Por su extensión, este anexo ha sido incluido en el CD adjunto a este trabajo de titulación.

ANEXO V

Capturas de pantalla de tablas de la base de datos

```
1 • SELECT * FROM tesis.ChallengingProblem;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

#	id	title	description	created_at	updated_at	Level_id
1	4	numeros pares de fibonacci	<h1>Números pares de Fibonacci</h1><p>Cada término en la secuencia de Fibo…<p><code>1, 2, 3, 5, 8, 13, 21, 34, 55, 89<…<p>Al …	2018-05-24 19:08:13	NULL	2
2	5	comparar numeros	<h1>Comparar datos</h1><p>Teniendo dos valores numéricos pasa… si el valor de <code>a</code> es men…	2018-05-24 22:02:19	NULL	1
3	6	palabras inversas	<h1>Palabras inversa.</h1><p>Dado un texto de longitud de palabras…<h3>Ejemplos</h3><h5>Ejemplo 1</h5><p>valor de entrada de tipo string</…<code>…	2018-05-25 08:42:21	NULL	3
*	NULL	NULL	NULL	NULL	NULL	NULL

Figura V.1 Tabla ChallengingProblema

```
1 •  SELECT * FROM tesis.DataType;
```

Result Grid | Filter Rows: | Edit: | Export/Import:

#	id	name	description	created_at	updated_at
1	1	boolean	type:boolean size:1 bit default:false	NULL	NULL
2	2	byte	type:byte size:8 bit 1 byte default:'\u000...	NULL	NULL
3	3	char	type:short size:16 bit 2 bytes default:0 ...	NULL	NULL
4	4	short	type:char size:16 bit 2 bytes default:0 ...	NULL	NULL
5	5	int	type:int size:32 bit 4 bytes default:0 ...	NULL	NULL
6	6	long	type:long size:64 bit 8 bytes default:0.L ...	NULL	NULL
7	7	float	type:float size:32 bit 4 bytes default:0.0f ...	NULL	NULL
8	8	double	type:double size:64 bit 8 bytes default:0.0...	NULL	NULL
9	9	string	''	NULL	NULL
*	NULL	NULL	NULL	NULL	NULL

Figura V.2 Tabla DataType

```
1 •  SELECT * FROM tesis.Language;
```

Result Grid | Filter Rows:

#	id	value	name	created_at	updated_at
1	1	""	""	NULL	NULL
2	2	CSharp	csharp	NULL	NULL
3	3	Java	java	NULL	NULL
*	NULL	NULL	NULL	NULL	NULL

Figura V.3 Tabla Language

```
1 SELECT * FROM tesis.Team;
```

Result Grid Filter Rows: Edit: Export/Import: Wrap Cell

#	id	name	workspace	User_id	Language_id	created_at	updated_at
1	6	TeamGoLang	1	3	3	2018-05-28 08:53:34	NULL
2	7	JavierMX	1	4	2	2018-05-28 08:54:01	NULL
3	8	XYZTeam	1	5	2	2018-05-28 08:54:36	NULL
4	9	asdfac	1	6	2	2018-05-28 08:54:56	NULL
5	10	AndruzDiaz	1	7	2	2018-05-28 08:56:03	NULL
6	11	ElvisBravo	1	8	2	2018-05-28 08:56:41	NULL
7	12	heavyvini	1	23	3	2018-05-28 08:57:33	NULL
8	13	devcode	1	9	2	2018-05-28 08:59:17	NULL
9	14	hernanreyes	1	10	2	2018-05-28 08:59:52	NULL
10	15	hiteam	1	11	2	2018-05-28 09:01:12	NULL
11	16	CNTteam	1	19	2	2018-05-28 09:03:19	NULL
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Figura V.4 Tabla Team

#	id	username	nickname	email	password	first_name	last_name	usertype
1	1	admin	admin@ad...	admin	$2a$14$/lQdTco77tdgkGhs.CM3uiyQLAE/EBlyiSir2heY8y3Ss7L/RWpm	admin	admin	admin
2	2	usuario1	usuario1	demo1@demo1.xyz	$2a$14$gXLe7QsHAh9lU38RmRd.ZaOFfKaA.FFtSz3ArCWtbCpz0zX9QBW	Usuario	Competidor	user
3	3	user-3	nickname-3	username3@live.com	$2a$14$gXLe7QsHAh9lU38RmRd.ZaOFfKaA.FFtSz3ArCWtbCpz0zX9QBW	Usuario	Prueba	user
4	4	user-4	nickname-4	username4@live.com	$2a$14$gXLe7QsHAh9lU38RmRd.ZaOFfKaA.FFtSz3ArCWtbCpz0zX9QBW	NULL	NULL	user
5	5	user-5	nickname-5	username5@live.com	$2a$14$gXLe7QsHAh9lU38RmRd.ZaOFfKaA.FFtSz3ArCWtbCpz0zX9QBW	NULL	NULL	user
6	6	user-6	nickname-6	username6@live.com	$2a$14$gXLe7QsHAh9lU38RmRd.ZaOFfKaA.FFtSz3ArCWtbCpz0zX9QBW	NULL	NULL	user
7	7	user-7	nickname-7	username7@live.com	$2a$14$gXLe7QsHAh9lU38RmRd.ZaOFfKaA.FFtSz3ArCWtbCpz0zX9QBW	NULL	NULL	user
8	8	user-8	nickname-8	username8@live.com	$2a$14$gXLe7QsHAh9lU38RmRd.ZaOFfKaA.FFtSz3ArCWtbCpz0zX9QBW	NULL	NULL	user
9	9	user-9	nickname-9	username9@live.com	$2a$14$gXLe7QsHAh9lU38RmRd.ZaOFfKaA.FFtSz3ArCWtbCpz0zX9QBW	NULL	NULL	user
10	10	user-10	nickname-10	username10@live.com	$2a$14$gXLe7QsHAh9lU38RmRd.ZaOFfKaA.FFtSz3ArCWtbCpz0zX9QBW	NULL	NULL	user
11	11	user-11	nickname-11	username11@live.com	$2a$14$gXLe7QsHAh9lU38RmRd.ZaOFfKaA.FFtSz3ArCWtbCpz0zX9QBW	NULL	NULL	user
12	12	09e2a22b-c...	oldmember	old@live.com	$2a$14$hsViKKhUv2lc7iSPOigL51fqUNE5JPHD2Ji67DVhnlxagDgv2a	Santiago	Alvarado	member
13	13	9aed224d-0...	9aed224d-0...	old1@live.com	$2a$14$aWbB4J8BXOod2leqDRSW6eEPdwAtQVgogxxi38MkL.6EkRMbw26my	NULL	NULL	member
14	14	27b22148-d...	27b22148-d...	hernanreyes7329@outlook.c	$2a$14$DYZRNZwauUtgON3LoEmhOc9lYbjFbafFl4EqOCk613Q8VHiC6i	NULL	NULL	user
15	15	be9279c2-a...	be9279c2-a...	paulgualotunadev@gmail.	$2a$14$6AEQKGnfNC6X/RlSzMyEe2pF/B6xKpygAWp0MsX1.yWtFnybMZu	NULL	NULL	user
16	16	0056b059-4...	0056b059-4...	andruzdiazaleman@gmail.co	$2a$14$H4jGjWM80mT5qckt7Qrh6uQFN8ZNUlQ3Y4Vq7y6WYoGAn2.KQpW1S	NULL	NULL	user
17	18	0fb35be7-5...	0fb35be7-5...	davip@live.com	$2a$14$Im959Q.r14pDBMBvVGL.GeanY2xoqXriRCb/0S1gbhQc3qplKM/7W	NULL	NULL	user
18	19	4f4c564c-4e...	4f4c564c-4...	elvisbravo262@hotmail.com	$2a$14$tAzZU1eilPYzJlinCurK2.fWJoqOBp0JO0cR5YgnjWugWpn.FXAUm	NULL	NULL	user
19	20	94603afa-e...	94603afa-e...	jose@gmail.com	$2a$14$voree7mNfsAhK9kbMsO83eTCqbjEldRMPDesGaG71HtSFlHKFS2G	NULL	NULL	user
20	22	0930ec8e-4...	0930ec8e-4...	anibal.cc.ch@gmail.com	$2a$14$C10BQJc377RCHBQVkQHoggeYmdGhLiHerQ56.n1Gb23VESKgSYvS	NULL	NULL	user
21	23	vochoa	pocho	heavyvinicio@hotmail.com	$2a$14$Vly7vpX5kmN6WsrNHdqs4uNhOKnWk.4lyhjsB8iOcqyHZ4KLH9Ui	Vinicio	Ochoa	user
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Figura V.5 Tabla User

ANEXO VI

Código

El código de la aplicación se encuentra en el CD adjunto

ORDEN DE EMPASTADO