UNIVERSIDAD MAYOR DE SAN ANDRÉS

FACULTAD DE CIENCIAS PURAS Y NATURALES

CARRERA DE INFORMÁTICA

TESIS DE GRADO

IMPLEMENTACIÓN DE MICROSERVICIOS PARA LA EVALUACIÓN DE PROGRAMAS BASADO EN CASOS DE PRUEBA

Tesis de Grado para obtener el Título de Licenciatura en Informática

Mención Ingeniería de Sistemas Informáticos

POR: DIEGO ADRIAN CHARCA FLORES

TUTOR METODOLÓGICO: M.SC. GROVER RODRIGUEZ RAMIREZ

ASESOR: M.SC. JORGE TERÁN POMIER

LA PAZ – BOLIVIA

2021

HOJA DE CALIFICACIONES

UNIVERSIDAD MAYOR DE SAN ANDRÉS

FACULTAD DE CIENCIAS PURAS Y NATURALES

CARRERA DE INFORMÁTICA

Tesis de Grado:

IMPLEMENTACIÓN DE MICROSERVICIOS PARA LA EVALUACIÓN DE PROGRAMAS BASADO EN CASOS DE PRUEBA

Presentado por: Diego Adrian Charca Flores

Para optar por el grado Académico de Licenciado en Informática

Mención Ingeniería de Sistemas Informáticos

Nota Numeral: 85

Nota Literal: Ochenta y cinco

Ha sido: Muy Buena

Director de la Carrera de Informática: Ph. D. Jose Maria Tapia Baltazar

Tutor: M.Sc. Grover Ramirez Rodriguez

Asesor: M.Sc. Jorge Teran Pomier

Tribunal: Ph.D. Yohoni Cuenca Sarzuri

Tribunal: Lic. Celia Tarquino Peralta

Tribunal: Lic. Victor Pozo Diaz

UNIVERSIDAD MAYOR DE SAN ANDRÉS
FACULTAD DE CIENCIAS PURAS Y NATURALES
CARRERA DE INFORMÁTICA

LA CARRERA DE INFORMÁTICA DE LA FACULTAD DE CIENCIAS PURAS Y NATURALES PERTENECIENTE A LA UNIVERSIDAD MAYOR DE SAN ANDRÉS AUTORIZA EL USO DE LA INFORMACIÓN CONTENIDA EN ESTE DOCUMENTO SI LOS PROPÓSITOS SON ESTRICTAMENTE ACADÉMICOS.

LICENCIA DE USO

El usuario está autorizado a:

a) visualizar el documento mediante el uso de un ordenador o dispositivo móvil.
b) copiar, almacenar o imprimir si ha de ser de uso exclusivamente personal y privado.
c) copiar textualmente parte(s) de su contenido mencionando la fuente y/o haciendo la referencia correspondiente respetando normas de redacción e investigación.

El usuario no puede publicar, distribuir o realizar emisión o exhibición alguna de este material, sin la autorización correspondiente.

DEDICATORIA

Dedico este trabajo a mi familia, quienes a lo largo de mi vida han velado por mi bienestar y educación siendo mi apoyo en todo momento.

A mis padres Macario y Eva, quienes me han dado todo lo que soy como persona, mis valores, mis principios, mi perseverancia y mi empeño, para seguir adelante.

A mi abuelo Pablo que me contagio de su alegría y me enseñó a siempre mostrar una sonrisa, aun en los momentos más difíciles.

AGRADECIMIENTOS

En primer lugar, quiero agradecer a Dios por darme la vida, fortaleza en los momentos difíciles y permitir que hoy concluya con una etapa más de mi vida.

Agradecer a mis padres por dar todo de ellos para que pueda salir adelante, por estar conmigo dándome todo su amor y apoyo incondicional.

Deseo expresar mi más profundo agradecimiento al M. Sc. Jorge Terán Pomier, por su apoyo, confianza y dirección como tutor, sobre todo por apoyar mi idea de tesis.

De igual manera agradecer al M. Sc. Grover Rodriguez Ramirez por su consejo, por su apoyo, por el tiempo dedicado, por la paciencia que tuvo, por las correcciones, observaciones y sugerencias que sirvieron para culminar satisfactoriamente este trabajo.

Para la Universidad Mayor de San Andrés que me dio la oportunidad de estudiar, y aprender a través de sus docentes, que, desde mi ingreso a esta casa superior de estudios, me formaron con sus enseñanzas y guía para afrontar esta vida profesional de ahora en adelante.

A mis amigos de la carrera con quienes compartimos muy buenos y malos momentos, que me apoyaron moralmente para seguir adelante.

diego.adrian.777@gmail.com

RESUMEN

Gracias a la arquitectura de microservicios se puede construir sistemas más escalables, mantenibles y eficientes, resolviendo en gran parte los problemas que conlleva la arquitectura monolítica.

El objetivo de este trabajo es implementar tales conocimientos en un juez virtual evaluador de programas basados en casos de prueba, para así poder satisfacer las necesidades que actualmente no puede cumplir el actual juez virtual con el que la Carrera de Informática actualmente cuenta.

Se utiliza la metodología de desarrollo Scrum, para la elaboración del sistema, detallando el proceso y los Sprints.

También se analiza las ventajas y desventajas al momento de implementar la arquitectura de microservicios a el presente trabajo.

Finalmente se presentan las conclusiones y recomendaciones que surgen después de la conclusión del trabajo de investigación.

Palabras clave: juez virtual, microservicios, ingeniería de software, caso de prueba, metodología SCRUM

Metodología: Metodología Ágil SCRUM

ABSTRACT

Thanks to the microservices architecture, more scalable, maintainable and efficient systems can be built, largely solving the problems that monolithic architecture entails.

The objective of this work is to implement such knowledge in a virtual judge evaluating programs based on test cases, in order to meet the needs that currently cannot be met by the current virtual judge that the Computer Science Degree of the Higher University of San Andrés currently has.

The Scrum development methodology is used, for the elaboration of the system, detailing the process and the Sprints.

The advantages and disadvantages when implementing the microservices architecture in this work are also analyzed.

Finally, the conclusions and recommendations that emerge after the conclusion of the research work are presented.

Keywords: online judge, microservices, software engineering, test cases, SCRUM Agile Methodology

Methodology: SCRUM Agile Methodology

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

CAPITULO I

MARCO REFERENCIAL

1.1 INTRODUCCIÓN

Con la llegada del internet se han construido muchas plataformas de educación virtual que ayudan de manera significativa en el proceso de aprendizaje de los estudiantes de distintas carreras. Estas plataformas permiten a docentes y estudiantes recibir y enviar tareas remotamente verificando si el estudiante inicio sesión y en qué tiempo la tarea fue recibida por el docente para verificar que la tarea fue entregada en el plazo determinado.

Los jueces virtuales o jueces en línea son plataformas de educación en la que los estudiantes pueden enviar soluciones de problemas de algoritmos computacionales.

En general, el propósito de los jueces virtuales es proveer una evaluación segura, confiable, y continua de problemas de algoritmos computacionales que los usuarios envían desde todas las partes del mundo. Durante la fase de envió, el código enviado se compila, si es necesario, y se verifica si se puede ejecutar con éxito en un entorno de evaluación homogéneo.

Después de la verificación, cada problema recibido por el juez virtual se evalúa de manera confiable con un conjunto de casos de prueba específicos para el problema. Para cada ejecución de un envió particular, se verifica que:

- El proceso de ejecución fue realizado sin errores.
- Cualquier limitación de recursos de un problema específico no fue excedida.
- El resultado de la ejecución cumple con las reglas descritas en la definición del problema.

En la Carrera de Informática de la Universidad Mayor de San Andrés se cuenta actualmente con un juez virtual que contribuye de manera parcial al aprendizaje de los estudiantes, ya que no satisface del todo las necesidades tanto como de los docentes, así como también de los estudiantes. Así también esta plataforma no tiene la flexibilidad para poder realizar cambios en su estructura ya que fue implementado de una manera monolítica.

Por lo tanto, el presente trabajo pretende desarrollar un juez virtual implementando por medio de microservicios y que evaluara los problemas con casos de prueba y que permitirá a

usuarios con los respectivos permisos crear problemas para que posteriormente otros usuarios puedan enviar la solución de dicho problema y conocer de manera instantánea el resultado de su solución. Los usuarios también podrán participar de competencias de programación.

1.2 PROBLEMA

1.2.1 ANTECEDENTE

La Carrera de Informática de la Universidad Mayor de San Andrés fue creada en 1974, como una mención del Departamento de Matemáticas; en 1984 se independiza administrativamente de éste, convirtiéndose posteriormente en Carrera de Informática con aproximadamente 300 alumnos.

Hace varios años se introdujo en la Carrera de Informática de la Universidad Mayor de San Andrés una modificación de un conocido juez virtual. Este juez actualmente es de mucha ayuda para el aprendizaje y la práctica de algoritmos, así como también sirve como plataforma para evaluar pruebas en las materias de algoritmia y programación.

Sin embargo, con una mayor utilización del juez han aumentado los requerimientos que no pueden satisfacerse con el actual juez virtual de la carrera de Informática. Este no está optimizado para recibirá más de 50 peticiones por segundo en el lado del servidor.

1.2.2 TRABAJOS SIMILARES

En la Carrera de informática de la Universidad Mayor de San Andrés los trabajos que se relacionan con la presente investigación son:

- Tutor inteligente para el proceso de aprendizaje en algoritmos y programación en el lenguaje Java, realizado por Jose Gonzalo Espejo Cuba bajo la modalidad de tesis de grado en el año 2015. El trabajo hace referencia a una plataforma para la enseñanza programación a estudiantes de la Carrera de Informática de la Universidad Mayor de San Andres.

En otras universidades:

- Diseño e implementación de un juez en línea para el desarrollo de competencias algorítmicas en la universidad libre, realizado por Ronald Fernando Chaparro Diaz bajo la modalidad de trabajo de grado en el año 2015 en la Universidad Libre de

Colombia. El trabajo define el diseño y la construcción de un Juez en Línea para el desarrollo de competencias algorítmicas.

1.2.3 PLANTEAMIENTO DEL PROBLEMA

La Carrera de Informática de la Universidad Mayor de San Andrés no cuenta con una plataforma moderna para la evaluación automática de programas basados en casos de prueba de manera rápida, segura, y fiable. Actualmente se cuenta con un juez virtual, pero este no se adapta a las necesidades de la población docente-estudiantil, ya que este es una modificación de un conocido juez virtual que se puede encontrar en internet. Por otro lado, el juez actual de la Carrera de Informática no es lo suficientemente optimo en cuanto a rendimiento se refiere, ya que no es capaz de recibir grandes peticiones en un corto periodo de tiempo. Siendo además la plataforma mencionada anteriormente un sistema monolítico haciendo difícil el mantenimiento y la mejora del sistema. De ahí, surge la necesidad de desarrollar un juez propio de la carrera que evalué programas basados en casos de prueba y que este implementado en microservicios que satisfaga las necesidades de la Carrera de Informática.

1.2.4 FORMULACIÓN DEL PROBLEMA

De la anterior sección nace nuestra pregunta:
¿Cómo se puede mejorar el rendimiento del juez virtual de la Carrera de Informática para la evaluación de programas basados en casos de prueba de manera automática, segura y rápida?

Se puede ver que existen muchos más problemas con respecto a la evaluación de problemas de programación y el juez virtual con el que actualmente cuenta la Carrera de Informática de la Universidad Mayor de San Andrés. Como los más relevantes tenemos:

- La evaluación de programas basados en casos de prueba de manera manual por lo general demora mucho y se debe hacer una prueba exhaustiva.
- Los estudiantes deben esperar un determinado tiempo para conocer el resultado de la solución que resuelve un problema específico.
- Dificultad al momento de detectar copia entre soluciones de problemas de programas basados en casos de prueba.

- No existe un repositorio catalogado de problemas de algoritmia computacional dentro de la Carrera de Informática.

1.3 OBJETIVOS

1.3.1 OBJETIVO GENERAL

Desarrollar un juez virtual mediante la implementación de microservicios, propio de la Carrera de Informática de la Universidad Mayor de San Andrés de alto rendimiento, escalable, y de fácil mantenimiento para la evaluación automática de programas basados en casos de prueba dirigida a estudiantes y docentes.

1.3.2 OBJETIVOS ESPECÍFICOS

- Evaluar de manera automática, rápida, segura y confiable programas basados en casos de prueba.
- Desarrollar juez virtual implementado por medio de microservicios propio de la Carrera de Informática, que contribuya a la educación de sus estudiantes.
- Mejorar el tiempo de respuesta del resultado de la solución con un veredicto pertinente.
- Crear un repositorio catalogado de problemas creados por la comunidad de la Carrera de Informática.

1.4 HIPÓTESIS

La implementación de microservicios para la evaluación de programas basados en casos de prueba **mejora** el rendimiento del juez virtual de la Carrera de Informática con una confiabilidad de **95%.**

1.5 JUSTIFICACIÓN

1.5.1 JUSTIFICACIÓN ECONÓMICA

Antes de la aparición del internet y de las tecnologías web el proceso de evaluación y prueba de problemas de programación se realizaba manualmente. Este proceso debido a la complejidad de algunas soluciones, en ocasiones demoraba demasiado.

Es por esto que se hace visible la necesidad de implementar un juez virtual implementado por medio de microservicios que evalúe automáticamente problemas de algoritmia computacional en Carrera de Informática de la Universidad Mayor de San Andrés, de manera segura, rápida y confiable.

De esta forma se ahorrará mucho tiempo al momento de revisar las soluciones de los problemas de algoritmia computacional basados en casos de prueba.

1.5.2 JUSTIFICACIÓN SOCIAL

La Carrera de Informática de la Universidad Mayor de San Andrés, necesita de recursos didácticos e innovadores que sirvan de apoyo en el proceso de aprendizaje de las materias de programación y algoritmia para así mejorar las habilidades en la resolución de algoritmos de los estudiantes.

1.5.3 JUSTIFICACIÓN TECNOLÓGICA

La presente propuesta se basa en el uso de una computadora con la cual el estudiante interactuara y enviara soluciones de problemas de programación a través de internet al juez virtual de evaluación automática de algoritmos computacionales que se pretende desarrollar. Además, se pretende mejorar el rendimiento mediante una arquitectura de microservicios, siendo esta muy utilizada en la actualidad.

1.5.4 JUSTIFICACIÓN CIENTÍFICA

La presente propuesta procesará los problemas de programación que será juzgados por el juez virtual de manera óptima ya que se hará uso de microservicios para cumplir con el objetivo. Así también se logrará gracias al uso de microservicios una alta escalabilidad y un fácil mantenimiento.

1.6 ALCANCES Y LIMITES

1.6.1 ALCANCES

La presente propuesta, pretende desarrollar un juez virtual para la práctica y el aprendizaje de algoritmos de programación, que nos permitirá mediante una arquitectura de microservicios:

- Crear problemas que serán resueltos con técnicas de algoritmia computacional.

- Evaluar de manera automática, rápida, segura y confiable problemas de algoritmia computacional.
- Publicar resultados de las soluciones enviadas por los usuarios de manera instantánea con el veredicto correspondiente.
- Crear concursos o pruebas de programación en el cual un grupo de estudiantes resolverán problemas y en el cual existirá uno o varios ganadores.
- Recibir grandes peticiones en el lado del servidor.

1.6.2 LÍMITES

- El juez virtual no contara con ningún tipo de comunicación entre usuarios dentro de la plataforma.
- No existirá una atención personalizada de parte del equipo de administración.
- El juez virtual inicialmente permitirá la evaluación de los problemas en los siguientes lenguajes de programación: Java, Python y C++.

1.6.3 ALCANCE GEOGRÁFICO

La ubicación geográfica del presente trabajo se lleva a cabo en la Zona Central de la ciudad de La Paz, en predios de la Carrera de Informática de la Universidad Mayor de San Andrés.

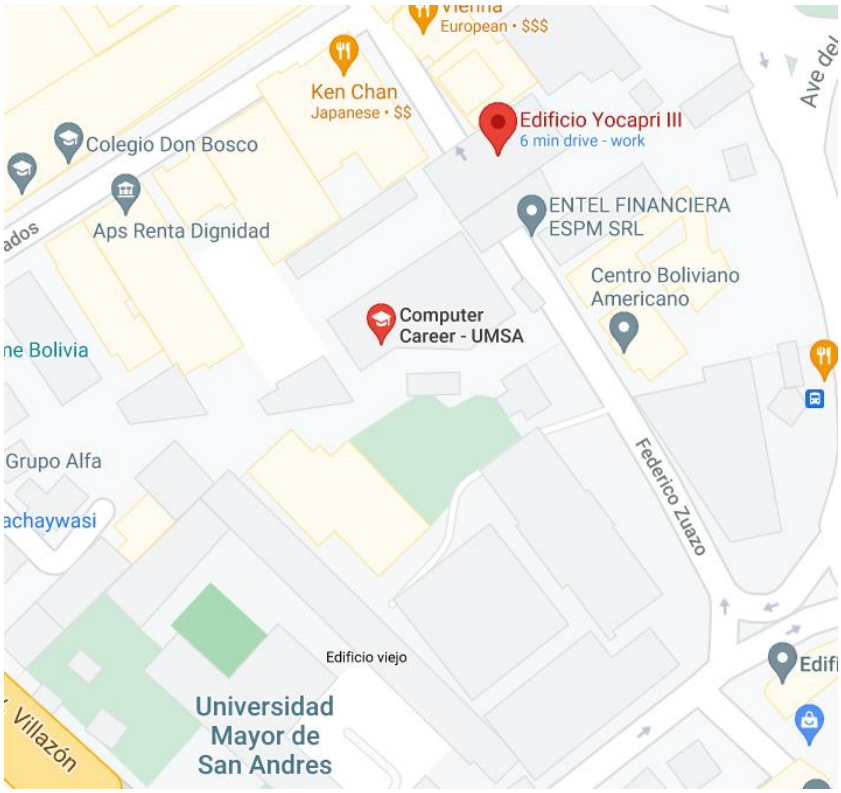


Figura 1 Ubicación de la Carrera de Informática

1.7 METODOLOGÍA

Se utiliza la metodología de investigación Sistémica ya que es uno de los instrumentos lógicos más contemporáneos en el ámbito de la metodología, orientado a la percepción holística, es decir, la totalidad de la realidad de donde se extraerá la propia problemática y las soluciones correspondientes. El Enfoque Sistémico (ES) es para la Informática una herramienta conceptual y de acción, interpreta lo concreto y facilita el pasaje de la teoría a la práctica, logrando mejores resultados. En el marco del Enfoque Sistémico, el análisis de sistemas, la modelización y la simulación se constituyen en las cuatro etapas fundamentales para el estudio del comportamiento dinámico de los sistemas complejos (Barchini, Sosa & Herrera, 2014). Las principales etapas de esta metodología son:

- Reconocimiento de problemas sistémicos: extracción de los aspectos relacionales y estructurales del problema.
- Abstracción del problema: identificación del problema dentro de un marco conceptual particular.

- Aplicación propia: uso de una herramienta metodológica apropiada para resolver el problema en su formulación abstracta.
- Interpretación de los resultados en términos del problema específico.

CAPITULO II

MARCO TEÓRICO

2.1 Arquitectura de Microservicios

> *"Una arquitectura de microservicios es un enfoque para desarrollar una aplicación software como una serie de pequeños servicios, cada uno ejecutándose de forma autónoma y comunicándose entre sí, por ejemplo, a través de peticiones HTTP a sus API.*
>
> *Normalmente hay un número mínimo de servicios que gestionan cosas comunes para los demás (como el acceso a base de datos), pero cada microservicio es pequeño y corresponde a un área de negocio de la aplicación.*
>
> *Además, cada uno es independiente y su código debe poder ser desplegado sin afectar a los demás. Incluso cada uno de ellos puede escribirse en un lenguaje de programación diferente, ya que solo exponen la API (una interfaz común, a la que le da igual el lenguaje de programación en la que el microservicio esté programado por debajo) al resto de microservicios."* (RedHat, 2021)

Entonces, cada servicio:

- Gestiona una parte de negocio única y exclusivamente
- Debe poder desplegarse y escalarse de forma independiente sin tener esto efecto sobre el resto de servicios
- Puede estar escrito con el lenguaje que mejor se adapte a su propósito, así como usar las tecnologías más convenientes (por ejemplo, bases de datos no-SQL cuando el resto de microservicios usan SQL).

2.1.1 Ventajas de la Arquitectura de Microservicios

Teniendo en cuenta la anterior descripción, no son pocas las ventajas que nos aporta esta arquitectura:

- Desarrollo modular: Cada servicio tiene una única meta, esto hace que su responsabilidad quede claramente acotada y su equipo de desarrollo puede especializarse en ello, sin interferencias de cambios en otros aspectos del negocio.

- Independencia de infraestructura: Los módulos pueden y deben seguir su propio ciclo de vida, desplegarse nuevas versiones cuando sea necesario (sin conflictos o necesidad de esperar a otros). Asimismo, cada microservicio es responsable de su propia infraestructura, incluyendo su propia base de datos.
- Escalado independiente: En un escenario donde tenemos un monolito y de repente tenemos un incremento bestial de un tipo de peticiones en concreto (registros por ejemplo, debido a una agresiva campaña de marketing), nuestra única solución va a ser escalar todo el proyecto a la vez, bien sea de forma vertical (añadiendo o mejorando el hardware) o si hemos sido suficientemente prudentes de tener separada la base de datos y otros servicios en otras máquinas, un escalado horizontal (aumentar el número de servers con servidores web y poner un balanceador de carga al frente). En un sistema basado en microservicios simplemente necesitas escalar de forma horizontal el número de servers que sirven el microservicio de usuarios.
- Optimización del esquema de datos: La independencia de infraestructura, permite que en nuestros modelos de datos guardemos únicamente lo imprescindible para lograr nuestro objetivo. Si por ejemplo disponemos de un microservicio de Posts y uno de Users, podemos guardar en la base de datos de Users tan solo un contador con el número de posts (si solo necesitamos ese dato), y en la base de datos de Posts nos podemos permitir almacenar solamente el id, nombre y apellidos de cada User, ya que no vamos a necesitar nada más en nuestras vistas. Resultado: prácticamente resolvemos todas las consultas a base de datos sin hacer JOIN, consultas directas a índice.
- Independencia de stack: Pueden desarrollarse con el lenguaje que mejor encaje en el problema a resolver, sin tener en cuenta en cuál (o cuáles) se ha desarrollado el resto de microservicios.

2.1.2 Desventajas de la Arquitectura de Microservicios

- Desarrollo más complejo: A medida que el número de microservicios crece, levantar un entorno de desarrollo para cada equipo es más complicado. Además, el *debug* de estas arquitecturas es bastante más complejo que el de un monolito.

- Versionado de microservicios: La independencia de cada microservicio puede llegar a ser un caos si no se versiona correctamente, ya que puede haber otros microservicios dependientes de uno que se acaba de actualizar que aún no sea compatible con los nuevos cambios, y eso nos obliga a mantener distintas versiones de cada microservicio en marcha que garantice el funcionamiento de todo el enjambre.
- Información duplicada: Gran parte de la información estará duplicada en las distintas bases de datos de los distintos microservicios, ya que cada uno debe de ser independiente a nivel datos.
- Consistencia eventual: Las actualizaciones pueden ser un quebradero de cabeza, por ejemplo, si el usuario decide actualizar su nombre y apellidos, ya que en el microservicio de Posts deberemos de actualizar este dato en todos sus posts. Esto puede dar lugar a problemas de consistencia eventual debido a la latencia entre comunicaciones (el dato se ha actualizado en algunos servicios, pero no en otros).

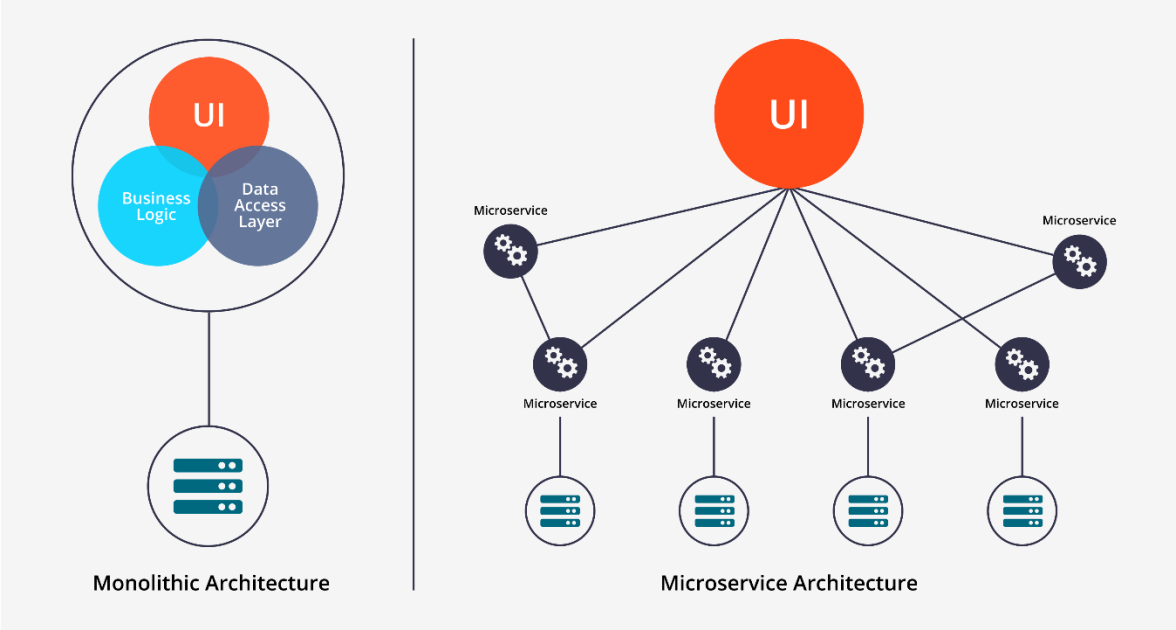


Figura 2. Arquitectura de Monolítica vs Arquitectura de Microservicios

2.1.3 Granularidad del Microservicio

Para diferenciar microservicios de servicios web tradicionales, es importante definir el tamaño que debe tener un microservicio para que se identifique como tal. Esta es una decisión de diseño relevante: si la separación es demasiado granular, entonces problemas identificados

en secciones anteriores pueden agravarse y degradar la calidad del sistema. Por otro lado, si los microservicios son demasiado grandes, el sistema se comporta de forma similar a un monolítico.

2.1.4 Formas de Comunicación de la Arquitectura de Microservicios

2.1.4.1 Microlitos sin infraestructura

Un caso muy común cuando se empieza con a migrar una arquitectura monolítica a una basada en microservicios es romper el monolito en otros más pequeños que popularmente reciben el nombre de microlitos.

El esquema que se detalla a continuación es el de Microlitos sin infraestructura: aunque se ha separado a nivel software el código en distintos servicios, se sigue compartiendo la infraestructura común (en este caso la base de datos).

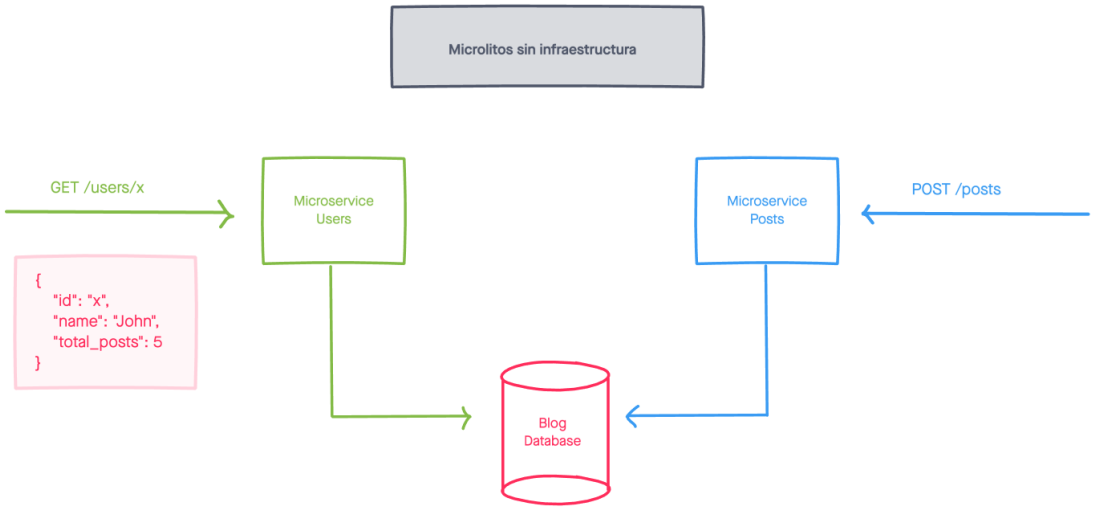


Figura 3 Microlitos sin infraestructura

Es muy difícil decir que cumple con todas las condiciones para ser un esquema de microservicios:

- Cada servicio gestiona una parte de negocio única y exclusivamente.
- No puede desplegarse y escalarse de forma independiente sin tener esto efecto sobre el resto de servicios. Si por ejemplo desplegamos una nueva versión del microservicio

de usuarios que aplica cambios a la base de datos, puede afectar al microservicio de posts.

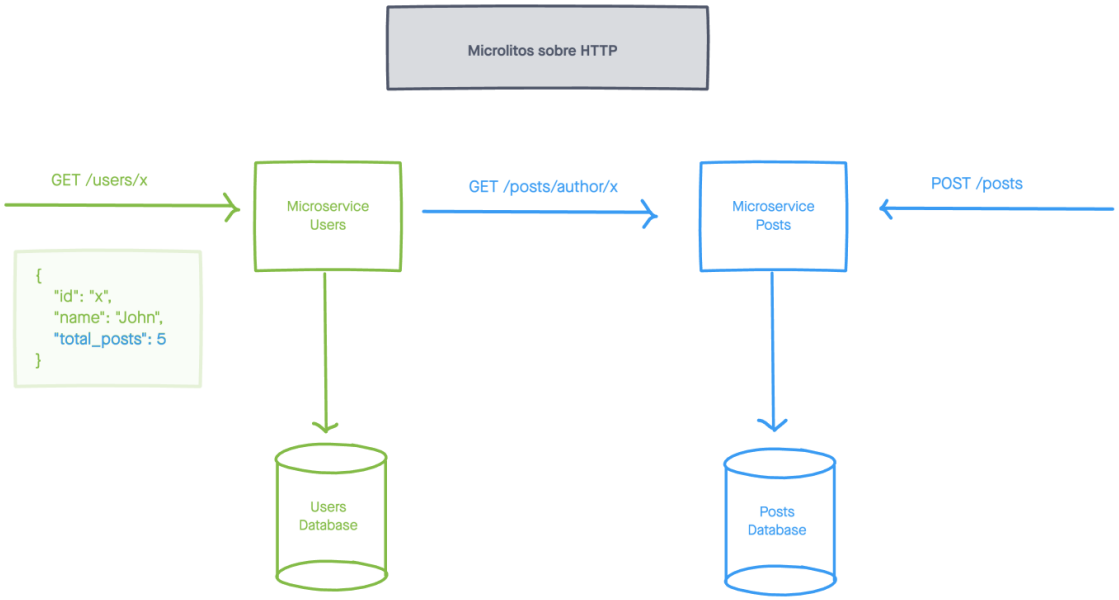
- No son independientes, una sobrecarga de peticiones en el microservicio de posts seguiría perjudicando al microservicio de usuarios.
- No utilizan mecanismos de comunicación de baja latencia para comunicarse, de hecho, no se comunican, la base de datos es el único intermediario, toda la información obtenida por cada microservicio sale del mismo sitio.
- Es un punto de partida, pero no podemos quedarnos aquí indefinidamente, debemos evolucionar nuestra arquitectura.

2.1.4.2 Microlitos vía HTTP

Una segunda parada en el camino hacia los microservicios independientes es hacer que los microlitos dejen de compartir la base de datos que usaron cuando estos formaban parte de una arquitectura monolítica.

Puesto que el microlito solo mantiene datos relacionados con la de su anterior base de datos, es posible que surja la necesidad de obtener alguna información relacionada con otras entidades.

Cada microlito expone una API por lo que parece una buena opción que nuestros microlitos se comuniquen vía HTTP para obtener la información necesaria.



2.2 Ingeniería de software

> *"La ingeniería de software es una aplicación del conocimiento, diseño y construcción de programas, los requerimientos de la tecnología de la información que demandan los individuos, negocios y gobiernos se hacen más complejos con cada año que pasa. En la actualidad, grandes equipos de personas crean programas de cómputo que antes eran elaborados por un solo individuo". (Wikipedia, Wikipedia - Ingenieria de Software, 2020)*

La ingeniería de software aplica diferentes normas y métodos que permiten obtener mejores resultados, en cuanto al desarrollo y uso del software, mediante la aplicación correcta de estos procedimientos se puede llegar a cumplir de manera satisfactoria con los objetivos fundamentales de la ingeniería de software.

Entre los objetivos de la ingeniería de software están:

- Mejorar el diseño de aplicaciones o software de tal modo que se adapten de mejor manera a las necesidades de las organizaciones o finalidades para las cuales fueron creadas.
- Promover mayor calidad al desarrollar aplicaciones complejas.
- Brindar mayor exactitud en los costos de proyectos y tiempo de desarrollo de estos.
- Aumentar la eficiencia de los sistemas al introducir procesos que permitan medir mediante normas específicas, la calidad del software desarrollado, buscando siempre la mejor calidad posible según las necesidades y resultados que se quieren generar.
- Una mejor organización de equipos de trabajo, en el área de desarrollo y mantenimiento de software.
- Detectar a través de pruebas, posibles mejoras para un mejor funcionamiento del software desarrollado.

2.3 Metodología de desarrollo de software

"La metodología de desarrollo de software es un proceso de software detallado y completo, es un conjunto de procedimientos, técnicas y ayuda a la documentación para el desarrollo de productos de software, además es un marco de trabajo usado para la estructura, planificación y control del proceso de desarrollo en un sistema de información. Una metodología debe definir con precisión los artefactos, roles y actividades involucrados, junto con prácticas y técnicas recomendadas, guías de adaptación de la metodología a proyectos, guías para uso de herramientas de apoyo.

El desarrollo de los sistemas tradicionales de ciclo de vida se originó en la década de 1960 para desarrollar a gran escala funcional de sistemas de negocio en una época de grandes conglomerados empresariales. La idea principal era continuar el desarrollo de los sistemas de información en una muy deliberada, estructurada y metódica, reiterando cada una de las etapas del ciclo de vida. Los sistemas de información en torno a las actividades resueltas pesadas para el procesamiento de datos y rutinas de cálculo". (WIkipedia, 2020)

2.4 Juez Virtual

Un juez en línea es un sistema en línea para probar programas en concursos de programación. También se utilizan para practicar para tales concursos. El sistema puede compilar, ejecutar y probar su código con datos preconstruidos (casos de prueba). El código enviado puede ejecutarse con restricciones, incluido el límite de tiempo, el límite de memoria, la restricción de seguridad, etc. El sistema capturará la salida del código y la comparará con la salida estándar. Luego, el sistema devolverá el resultado.

Cuando se encuentran errores en una salida de resultados standard, el envío será juzgado como no exitoso. El usuario deberá corregir los errores en el código, y re-enviar el código una vez más para ser re-juzgado.

Algunos ejemplos de jueces en línea son:

- El Juez de la Universidad de Valladolid, con cerca de 2000 problemas, con soporte para C, C++, Pasca y Java.

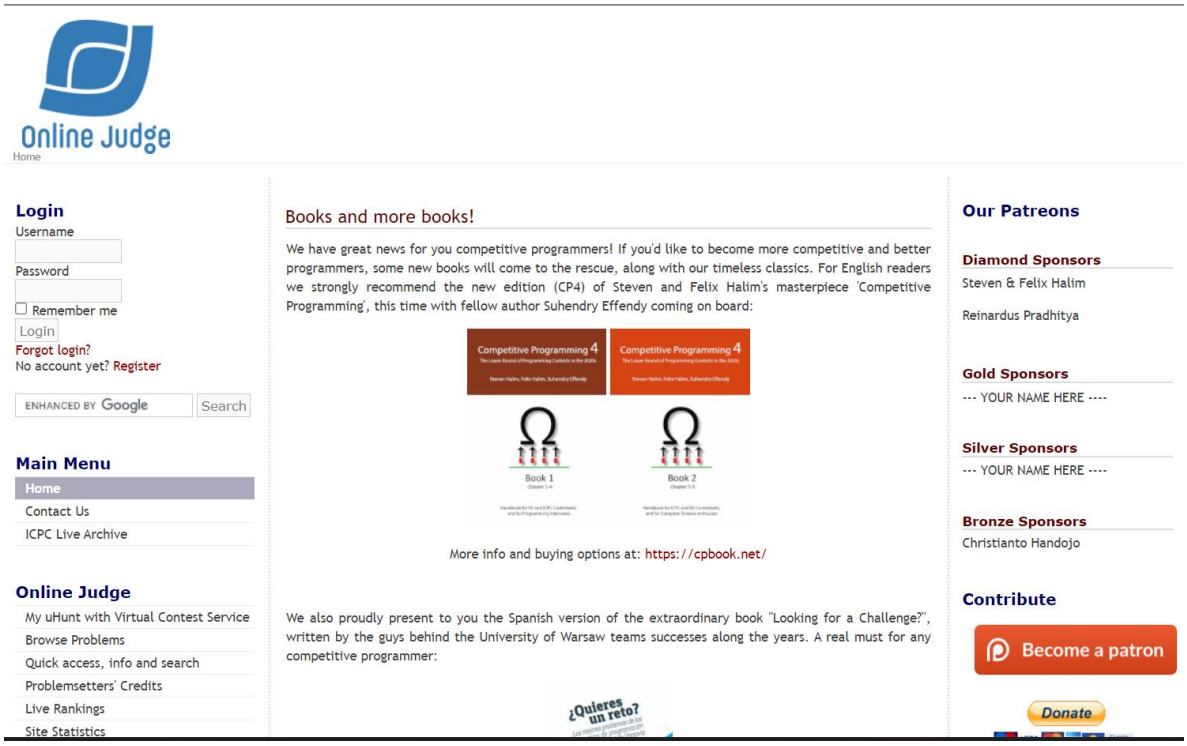


Figura 4 Página principal del Juez de la UVA(UVA, 2021)

- Sphere Online Judge, con cerca de 3000 problemas formulados en inglés, portugués, y español con soporte para 30 diferentes lenguajes de programación.

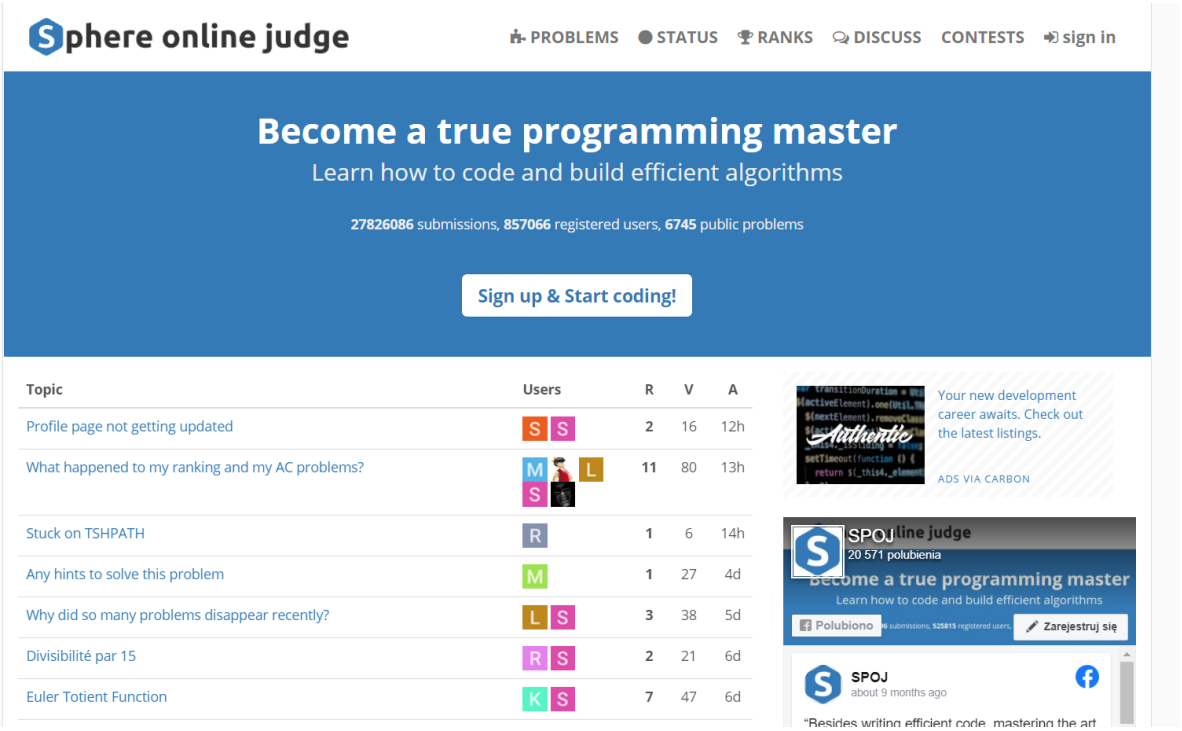


Figura 5. Página principal del juez SPOJ (SPOJ, 2021)

- Codeforces

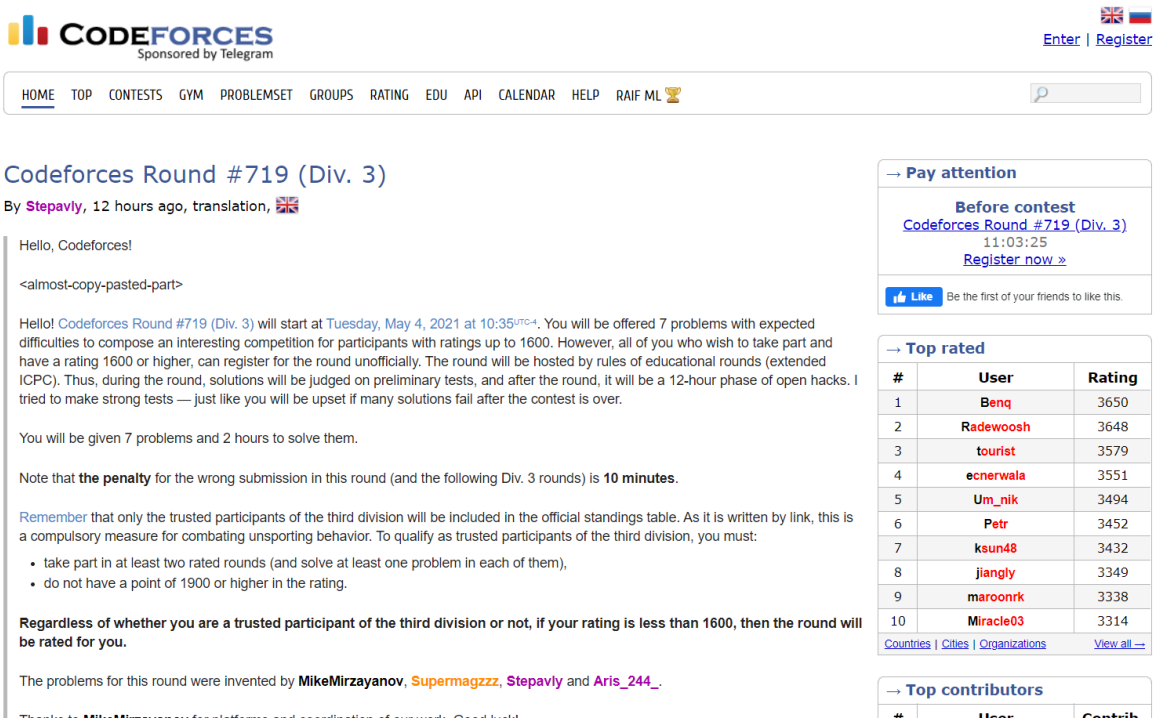


Figura 6. Página principal del juez en linea Codeforces (Codeforces, 2021)

- Codechef

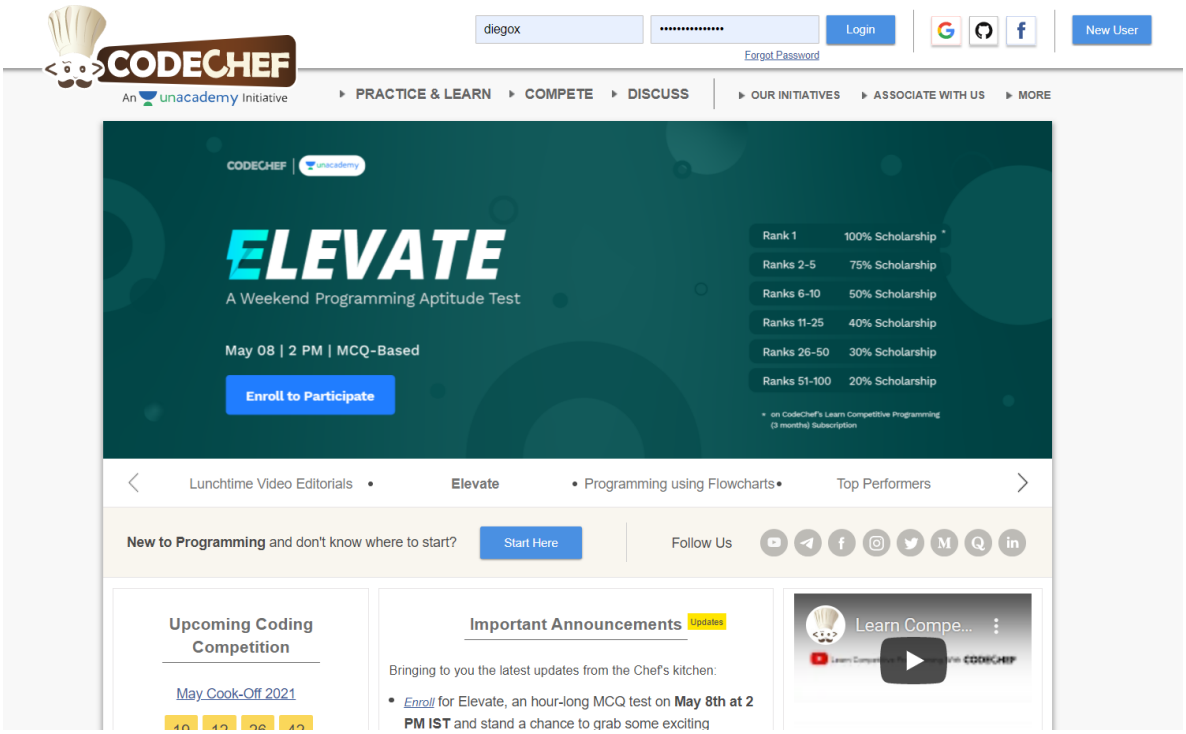


Figura 7. Página principal del juez en linea Codechef (Codechef, 2021)

2.5 Ingeniería web

La ingeniería web no es un nuevo paradigma o un nuevo tipo de ingeniería, los métodos de desarrollo web toman y especializan aquellas técnicas de la IS (Information System – Sistema de Información) más útiles para el caso concreto del software web.

El desarrollo de aplicaciones Web posee determinadas características que lo hacen diferente del desarrollo de aplicaciones o software tradicional y sistemas de información. La ingeniería de la Web es multidisciplinar y aglutina contribuciones de diferentes áreas: arquitectura de la información, ingeniería de hipermedia/hipertexto, ingeniería de requisitos, diseño de interfaz de usuario, usabilidad, diseño gráfico y de presentación, diseño y análisis de sistemas, ingeniería de software, ingeniería de datos, indexado y recuperación de información, testeo, modelado y simulación, despliegue de aplicaciones, operación de sistemas y gestión de proyectos.

La ingeniería de la Web no es un clon o subconjunto de la ingeniería de software, aunque ambas incluyen desarrollo de software y programación, pues a pesar de que la ingeniería de

la Web utiliza principios de ingeniería de software, incluye nuevos enfoques, metodologías, herramientas, técnicas, guías y patrones para cubrir los requisitos únicos de las aplicaciones web. Sin embargo, el término de ingeniería de la web ha sido un término muy controvertido especialmente para profesionales en disciplinas tales como la ingeniería del software ya que no la consideran como un campo dentro de la ingeniería.

2.6 React JS

React (también llamada React.js o ReactJS) es una biblioteca Javascript de código abierto diseñada para crear interfaces de usuario con el objetivo de facilitar el desarrollo de aplicaciones en una sola página. Es mantenido por Facebook y la comunidad de software libre, han participado en el proyecto más de mil desarrolladores diferentes.

React intenta ayudar a los desarrolladores a construir aplicaciones que usan datos que cambian todo el tiempo. Su objetivo es ser sencillo, declarativo y fácil de combinar. React sólo maneja la interfaz de usuario en una aplicación; React es la Vista en un contexto en el que se use el patrón MVC (Modelo-Vista-Controlador) o MVVM (Modelo-vista-modelo de vista). También puede ser utilizado con las extensiones de React-based que se encargan de las partes no-UI (que no forman parte de la interfaz de usuario) de una aplicación web.

Según el servicio de análisis Javascript (en inglés "javascript analytics service"), Libscore, React actualmente está siendo utilizado en las páginas principales de Imgur, Bleacher Informe, Feedly, Airbnb, SeatGeek, HelloSign, y otras.

2.7 NodeJS

Node.js es un entorno en tiempo de ejecución multiplataforma, de código abierto, para la capa del servidor (pero no limitándose a ello) basado en el lenguaje de programación ECMAScript, asíncrono, con I/O de datos en una arquitectura orientada a eventosy basado en el motor V8 de Google. Fue creado con el enfoque de ser útil en la creación de programas de red altamente escalables, como, por ejemplo, servidores web.4 Fue creado por Ryan Dahl en 2009 y su evolución está apadrinada por la empresa Joyent, que además tiene contratado a Dahl en plantilla.

Gracias al enfoque de desarrollo basado en pequeños servicios de forma autónoma y comunicándose entre sí, las empresas pueden tener arquitecturas sólidas diferenciadas sobre

las antiguas monolíticas. La gran diferentcia es que en las monolíticas todos los servicios se desplegaban en todos los servidores, pero con los microservicios se pretenden distribuir/desplegar estos servicios entre los servidores de forma desacoplada, replicando según se desee.

Node.js ha ganado mucha tracción en el entorno empresarial: desde LinkedIn a Walmart, o en España Zara o ING Direct, han migrado sus sistemas a Node.js. Su facilidad de uso, la programación en un lenguaje muy común como es JavaScript y su buen rendimiento lo han situado a la cabeza del desarrollo de aplicaciones web.

2.8 Docker

> *"Docker es un proyecto de código abierto que automatiza el despliegue de aplicaciones dentro de contenedores de software, proporcionando una capa adicional de abstracción y automatización de virtualización de aplicaciones en múltiples sistemas operativos. Docker utiliza características de aislamiento de recursos del kernel Linux, tales como cgroups y espacios de nombres (namespaces) para permitir que "contenedores" independientes se ejecuten dentro de una sola instancia de Linux, evitando la sobrecarga de iniciar y mantener máquinas virtuales." (Wikipedia, WIkipedia, Docker, 2020)*
>
> *El potente concepto de Microservicios está cambiando poco a poco la industria. Grandes servicios monolíticos están dando paso lentamente al enjambre microservicios pequeños y autónomos que trabajan en conjunto. El proceso va acompañado de otra tendencia del mercado: la contenerización. Juntos, ayudan a construir sistemas sin precedentes. La contenerización cambia no sólo la arquitectura de los servicios, sino también la estructura de ambientes utilizados para crearlos. (GuiaDev, 2020)*

Ahora, cuando el software se distribuye en contenedores, los desarrolladores tienen plena libertad para decidir qué aplicaciones necesitan. Como resultado, incluso los entornos complejos, como los servidores de grades bases de datos e infraestructura de análisis complejos pueden crear instancias en cuestión de segundos. El desarrollo de software se hace más fácil y más eficaz.

2.9 PostgreSQL

> *"PostgreSQL es un sistema de gestión de bases de datos relacional orientado a objetos y de código abierto, publicado bajo la licencia PostgreSQL,1 similar a la BSD o la MIT.*
>
> *Como muchos otros proyectos de código abierto, el desarrollo de PostgreSQL no es manejado por una empresa o persona, sino que es dirigido por una comunidad de desarrolladores que trabajan de forma desinteresada, altruista, libre o apoyados por organizaciones comerciales. Dicha comunidad es denominada el PGDG". (*Group, 2020*)*

2.10 Metodología de Desarrollo Scrum

Una buena práctica de organización de proyectos es hacer uso de una metodología de desarrollo para poder así organizar al equipo, los tiempos y recursos. En la actualidad por el gran nivel competente entre empresas es necesario producir software de la manera más rápida posible, de esta forma es posible obtener una retro alimentación de los usuarios para poder realizar mejoras.

> *En la actualidad, las diferentes metodologías ágiles ganan popularidad entre empresas al permitir la entrega constante de productos funcionales. La metodología Scrum fue desarrollada por Ken Schwaber y Jeff Shuterland en los años 90 Scrum es un marco de trabajo utilizado para la construcción de productos complejos. Scrum al no ser un proceso o técnica, pero este cuenta con diferentes procesos y técnicas. Scrum muestra de forma clara la relación eficaz entre la administración de un producto y las buenas prácticas de desarrollo. (Schwaber, 2016)*

Scrum como se observa en la Figura 7, es un marco de trabajo con el cual es posible trabajar con problemas complejos con los que lidian diferentes equipos haciendo entrega de productos de valor alto de una forma productiva y creativa. Scrum al ser un marco de trabajo ágil, sigue los principios planteados en el manifiesto ágil. La Tabla 1 muestra estos principios. (Beck, et al., 2001)

Tabla 1: Principio de Manifiesto Ágil

Principio de Manifesto Ágil

Nro.	**Manifiesto Ágil**
1	Satisfacer al cliente mediante la entrega temprana y continua de software con valor.
2	Es aceptable que los requisitos cambien, incluso en etapas tardías de desarrollo. Los procesos ágiles aprovechan el cambio para proporcionar una ventaja competitiva al cliente.
3	Entrega frecuente de software funcional, entre dos semanas y dos meses, con preferencia al periodo de tiempo más corto possible.
4	Los responsables de negocio y desarrolladores trabajan juntos de forma cotidiana durante todo el proyecto
5	Los proyectos se desarrollan en torno a individuos motivados. Es necesario brindarles el entorno y apoyo que estos requieren así confiarles la ejecución del trabajo
6	El método más eficiente y efectivo de comunicar información al equipo de desarrollo y entre sus miembros es la conversación cara a cara
7	El software funcional es la medida principal de progreso
8	Los procesos ágiles promueven el desarrollo sostenible. Los promotores, desarrolladores y usuarios deben ser capaces de mantener un ritmo constante de forma indefinida
9	La atención continua a la excelencia técnica y al buen diseño mejora la agilidad
10	La simplicidad, o el arte de maximizar la cantidad de trabajo no realizado es esencial

11	Las mejores arquitecturas, requisitos y diseños surgen de equipos auto-organizados
12	A intervalos regulares el equipo reflexiona sobre cómo ser más efectivo para poder así ajustar y perfeccionar su comportamiento en consecuencia.

Nota. Fuente: *sinnaps.com, Metodología Scrum*

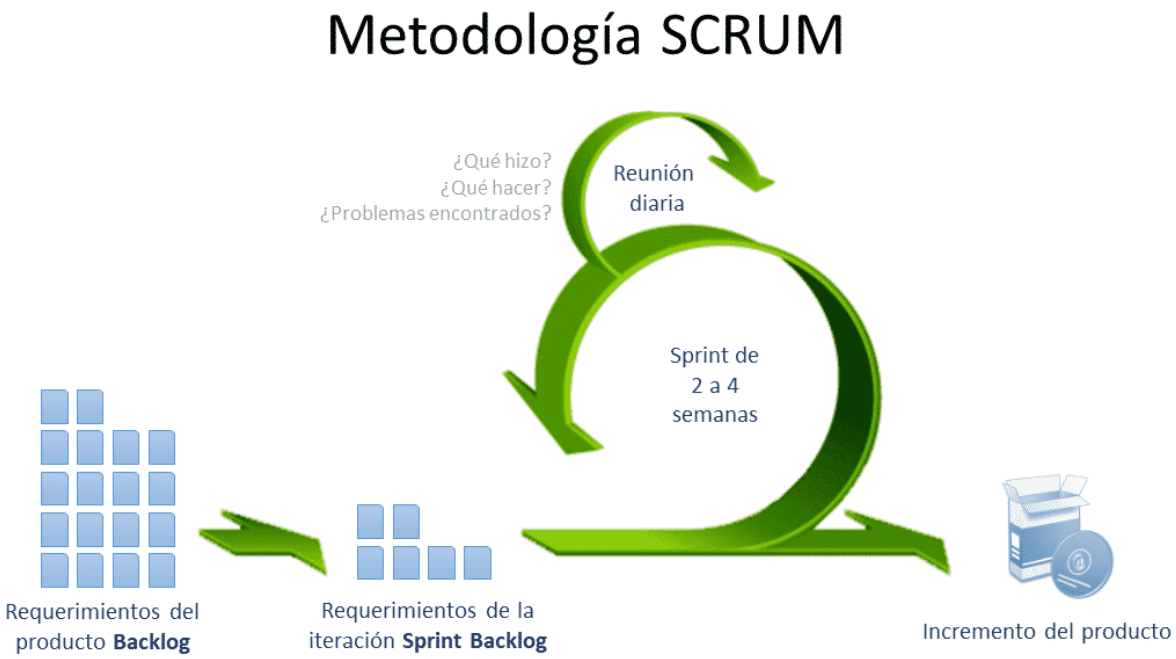


Figura 8. Etapas de la Metodología Scrum

2.10.1 Roles de Scrum

El Equipo Scrum consiste en un Product Owner, Development Team y un Scrum Master. Los Equipos Scrum son auto organizados y multifuncionales. Los equipos auto organizados eligen la mejor forma de llevar a cabo su trabajo y no son dirigidos por personas externas al equipo. Los equipos multifuncionales tienen todas las competencias necesarias para llevar a cabo el trabajo sin depender de otras personas que no son parte del equipo. El modelo de equipo en Scrum está diseñado para optimizar la flexibilidad, la creatividad y la productividad. Los Equipos Scrum entregan productos de forma iterativa e incremental, maximizando las oportunidades de obtener retroalimentación. Las entregas incrementales de

producto “Terminado” aseguran que siempre estará disponible una versión potencialmente útil y funcional del producto.

Los Equipos Scrum entregan productos de forma iterativa e incremental, maximizando las oportunidades de obtener retroalimentación. Las entregas incrementales de producto “Terminado” aseguran que siempre estará disponible una versión potencialmente útil y funcional del producto.

Product Owner, este miembro es el responsable de maximizar el valor del producto y el trabajo del Equipo de Desarrollo. El Product Owner es el único miembro responsable del backlog del producto. Por lo general este miembro es una sola persona y no un comité, sin embargo, este se encarga de plasmar los requerimientos de algún comité en el backlog del producto. Las decisiones realizadas por el Product Owner se hacen visibles en el backlog del producto. De esta forma, nadie más que el Product Owner puede ordenar que el Equipo de Desarrollo trabaje en otro tipo de requerimientos, y el Equipo de desarrollo no tiene permitido en trabajar en base a que otros digan.

Development Team, es el Equipo de Desarrollo consiste de profesionales que realizan el trabajo de realizar entregas de un producto hecho al final de cada sprint o “iteración”. Un Equipo de Desarrollo es auto-organizado y funcional para poder así maximizar la efectividad del equipo.

Por lo general un equipo de desarrollo debe estar formado entre tres y nueve elementos, de esta forma no tener conflictos de organización ni minimizar los resultados en cada sprint.

Scrum Master, es responsable de asegurar que todas las actividades de Scrum se lleven de forma correcta. Es necesario que este rol muestre y haga entender la teoría, reglas y prácticas de Scrum. Es necesario que el Scrum Master ayude a los elementos fuera del equipo a entender cuáles son las interacciones con el equipo son útiles y cuáles no, de este modo el Scrum Master ayudará a modificar y adaptar estas interacciones para poder maximizar el valor de todo el equipo en conjunto.

2.10.2 Eventos de Scrum

En Scrum existen eventos predefinidos con el fin de crear regularidad y minimizar la necesidad de reuniones no definidas en Scrum. Todos los eventos son bloques de tiempo, de

tal modo que todos tienen una duración máxima. Una vez que comienza un Sprint, su duración es fija y no puede acortarse o alargarse. Los demás eventos pueden terminar siempre que se alcance el objetivo del evento, asegurando que se emplee una cantidad apropiada de tiempo sin permitir desperdicio en el proceso.

Además del propio Sprint, que es un contenedor del resto de eventos, cada uno de los eventos de Scrum constituye una oportunidad formal para la inspección y adaptación de algún aspecto.

Estos eventos se diseñaron específicamente para habilitar los pilares vitales de transparencia e inspección. La falta de alguno de estos eventos da como resultado una reducción de la transparencia y constituye una oportunidad perdida de inspección y adaptación.

Sprint – El corazón de Scrum se yace en el Sprint o iteración, la cual cuenta con un lapso de tiempo menor o igual a un mes en el cual es necesario presentar una versión incrementada, utilizable, con el potencial de ser el producto final en base a una meta inicial. Un Sprint es consistente por los esfuerzos de desarrollo empleados. Un Sprint inicia al finalizar un Sprint anterior. Un Sprint consiste de la Planificación de Sprint, Daily Scrum, Revisión del Sprint y la Retrospectiva del Sprint.

Planificación de Sprint – Este evento realiza la planificación de todas las metas a cumplirse durante el Sprint, la duración de esta planificación debe ser menor o igual a ocho horas, sin embargo, la duración de la planificación es directamente proporcional a la duración del Sprint, es decir, la planificación de un Sprint corto debe ser corta. En este evento el Scrum Master debe encargarse de que el equipo entienda el propósito del Sprint a realizarse.

La meta del Sprint es un conjunto de objetivos que pueden hacerse gracias al backlog del producto. La meta es una guía para el equipo de desarrollo para la auto-organización de este. Estos objetivos son un conjunto de historias de usuario a desarrollarse que se obtienen del backlog del producto a las cuales se asignan tareas, puntos de complejidad, encargados que realizaran la historia de usuario.

Daily Scrum – Es una reunión que se realiza todos los días con una duración máxima de quince minutos en la que todos los miembros del equipo tocan los siguientes puntos:

- Que se realizó el día anterior.

- Que se realizará en la jornada presente.
- La existencia de algún elemento que bloquee al miembro del equipo.

Revisión del Sprint – Este evento se realiza al haber finalizado el Sprint para poder inspeccionar las mejoras en el producto trabajado y verificar los cambios que se realizaron al backlog del producto. Para un Sprint de un mes se recomienda que este evento dure a lo mucho cuatro horas.

Los diferentes miembros del equipo comparten los avances que se realizaron durante el Sprint con las partes interesadas en el proyecto, por lo general clientes que son invitados previamente por el Product Owner.

El Product Owner debe los elementos del backlog del producto que fueron realizados, basados en eso, el Equipo de Desarrollo demuestra el trabajo que realizó y responde a las preguntas respecto al trabajo realizado en el Sprint actual. Es posible que el mercado al que apunta la aplicación haya cambiado, es por ello que en este evento es necesario aclarar este tipo de tópicos para poder aumentar el valor de la siguiente iteración. En general, el grupo completo colabora de tal forma que se consiguen puntos a tocar en la planificación del siguiente Sprint que pueda aumentar el valor del producto.

Retrospectiva del Sprint – Este evento permite que el equipo encuentre falencias o propuestas para mejorar el siguiente Sprint. Este evento se lleva a cabo tras haber concluido la revisión del Sprint y antes de comenzar la planificación del siguiente Sprint. Se recomienda que la duración de este evento sea a lo mucho de tres horas para sprints de un mes.

Durante este evento el equipo debe planificar diferentes formas para poder incrementar la calidad del producto. Para ello es necesario inspeccionar como fueron las personas del equipo, relaciones y uso de las herramientas de tal forma que sea posible identificar falencias o mejoras.

De este modo, al finalizar la Retrospectiva del Sprint el equipo habrá encontrado mejoras que puedan ser aplicadas al siguiente Sprint. No es obligatorio que estas mejoras sean encontradas y aplicadas durante y después de la Retrospectiva del Sprint, estas mejoras pueden ser aplicadas en cualquier momento de un Sprint.

2.10.3 Artefactos de Scrum

Los artefactos de Scrum representan trabajo o valor en diversas formas que son útiles para proporcionar transparencia y oportunidades para la inspección y adaptación. Los artefactos definidos por Scrum están diseñados específicamente para maximizar la transparencia de la información clave, necesaria para asegurar que todos tengan el mismo entendimiento del artefacto.

Backlog del Producto – El Backlog del Producto es una lista los requerimientos, funciones, características, mejoras y arreglos necesarios para el producto, además de ser la única fuente de requerimientos al cual pueden realizarse cambios. El Product Owner es el encargado de mantener y gestionar este artefacto incluyendo su contenido, disponibilidad y orden.

Al ser Scrum una metodología iterativa, el backlog del producto nunca se encontrará finalizada, mientras exista el producto existirá el backlog del producto. Inicialmente el contenido de este artefacto solo muestra los requerimientos que fueron comprendidos. Este artefacto evoluciona a medida que el producto y el entorno en el que el producto funcionará evolucionan, es decir, cualquier cambio en requerimientos de negocio o condiciones de mercado puede causar cambios en el backlog del producto.

Es necesario refinar el backlog del producto adicionando estimados, detalles y orden a los elementos que yacen en este. Durante este proceso en el que trabajan el Product Owner y el Equipo de desarrolla los elementos del backlog del producto son revisados.

Backlog del Sprint – El backlog del sprint es un conjunto de elementos del backlog del producto que son elegidos para llevarse a cabo durante un Sprint, además de tener un plan para entregar la mejora del producto y cumplir la meta del Sprint. El backlog del sprint es un pronóstico del equipo de desarrollo sobre que funcionalidad tendrá la siguiente mejora del producto y el trabajo necesario para poder finalizar dichas funcionalidades. Este artefacto hace visible todo el trabajo necesario identificado por el Equipo de Desarrollo para cumplir la meta del Sprint.

A medida que más trabajo es necesario, es necesario adicionarlo al backlog del sprint. Por otro lado, a medida que se realice el trabajo o se lo complete, el trabajo restante estimado se actualiza. A diferencia del backlog del producto, el backlog del sprint solo es administrado

por el Equipo de Desarrollo, de tal forma que solo este equipo puede realizar cambios en este artefacto, adicionar elementos necesarios o remover elementos innecesarios.

Incremento o mejora del producto – Al final un sprint el incremento o mejora del producto es el total de todos los elementos completados que pertenecen al backlog del sprint y el valor de las mejoras del producto de sprints anteriores. Al finalizar un sprint un nuevo incremento está hecho lo que significa que este incremento debe encontrarse en una condición útil independientemente que el Producto Owner decida que el incremento se publicará o no.

CAPITULO III

DISEÑO METODOLÓGICO

3.1 INTRODUCCIÓN

El desarrollo del sistema basado en microservicios fue desarrollado haciendo uso de la metodología de desarrollo Scrum. En este capítulo se muestra la aplicación de esta metodología de desarrollo en sus diferentes etapas, tales como: Backlog, Historia de Usuario, Tareas y Sprints.

3.2 BACKLOG INICIAL DE HISTORIAS DE USUARIO

La tabla muestra el backlog inicial de las Historias de Usuario para el sistema de microservicios por orden de importancia

Backlog inicial de Historias de Usuario	
Nro.	**Nombre de la Historia de Usuario**
1	Identificación de tecnologías
2	Diseño de la base de datos
3	Implementación del servicio principal
4	Implementación de la base de datos
5	Implementación del servicio evaluador de programas
6	Diseño inicial de las pantallas de la interfaz de usuario
7	Implementación de la interfaz de usuario
8	Integración de los microservicios
9	Despliegue del sistema basado en microservicios

Nota. Fuente: Elaboración propia

Tabla 2: Backlog Inicial

3.3 DESCRIPCIÓN DE LAS HISTORIAS DE USUARIO

Para poder lograr los objetivos planteados en el Capítulo I, es necesario tener las historias de usuario para así poder en claro lo que se debe realizar.

Como punto de partida se identificaron 9 Historias de Usuario, divididas con sus respectivas categorías, y las cuales se describen en las siguientes tablas

Historia de usuario		
ID: 1	**Nombre**: Identificación de Tecnologías	
	Usuario: Equipo de Desarrollo	**Categoría**: Ninguna
Descripción: Identificación de tecnologías para la realización del prototipo en cuanto a hardware y software.		
Nota. Fuente: Elaboración propia		

Tabla 3: Historia de Usuario 1

Historia de usuario		
ID: 2	**Nombre**: Diseño de la base de datos	
	Usuario: Equipo de Desarrollo	**Categoría**: Base de Datos
Descripción: Diseño para inicial de la base de datos con su respectivo diagrama		
Nota. Fuente: Elaboración propia		

Tabla 4: Historia de Usuario 2

Historia de usuario		
ID: 3	**Nombre**: Implementación del servicio principal	
	Usuario: Equipo de Desarrollo	**Categoría**: Microservicio
Descripción: Implementar un servicio de backend para la autentificación de usuarios, manejo de problemas, manejo de concursos de programación y reporte de resultados.		
Nota. Fuente: Elaboración propia		

Tabla 5: Historia de Usuario 3

Historia de usuario		
ID: 4	**Nombre**: Implementación de la base de datos	
	Usuario: Equipo de Desarrollo	**Categoría**: Base de Datos
Descripción: Con la implementación inicial del servicio principal, se procede a implementar el diseño de base de datos con el ORM Sequelize.		
Nota. Fuente: Elaboración propia		

Tabla 6: Historia de Usuario 4

Se puede encontrar un ejemplo de implementación de la Historia de usuario 4 en el Anexo A.

Historia de usuario		
ID: 5	**Nombre**: Implementación del servicio evaluador de programas	
	Usuario: Equipo de Desarrollo	**Categoría**: Microservicio

Descripción: Implementar un microservicio que se encargue de la evaluación de programas, este deberá recibir las configuraciones y los casos de prueba y deberá retornar el correspondiente veredicto.
Nota. Fuente: Elaboración propia

Tabla 7: Historia de Usuario 5

Se puede encontrar un ejemplo de implementación de la Historia de usuario 5 en el Anexo B.

Historia de usuario		
ID: 6	**Nombre**: Diseño inicial de las pantallas de la interfaz de usuario	
	Usuario: Equipo de Desarrollo	**Categoría**: Diseño
Descripción: Diseñar bocetos para las distintas páginas de la interfaz de usuario.		
Nota. Fuente: Elaboración propia		

Tabla 8: Historia de Usuario 6

Historia de usuario		
ID: 7	**Nombre**: Implementación de la interfaz de usuario	
	Usuario: Equipo de Desarrollo	**Categoría**: Microservicio
Descripción: Implementar las paginas basados en los bocetos disonados anteriormente haciendo uso de tecnologías para el desarrollo web: ReactJS, MaterialUI.		
Nota. Fuente: Elaboración propia		

Tabla 9: Historia de Usuario 7

Historia de usuario		
ID: 8	**Nombre**: Integración de los microservicios	
	Usuario: Equipo de Desarrollo	**Categoría**: Microservicio
Descripción: Con la implementación inicial de los microservicios se deberá integrar estos haciendo uso de sus respectivas RESTful API's.		
Nota. Fuente: Elaboración propia		

Tabla 10: Historia de Usuario 8

Historia de usuario		
ID: 9	**Nombre**: Despliegue del sistema basado en microservicios	
	Usuario: Equipo de Desarrollo	**Categoría**: Microservicio
Descripción: Una realizada la integración de los microservicios se deberá desplegar el sistema a un servidor para poder realizar las pruebas correspondientes.		
Nota. Fuente: Elaboración propia		

Tabla 11: Historia de Usuario 9

3.4 SPRINTS

En esta sección se planifico los Scrum Sprints, los cuales ayudan a completar los alcances que se planearon en el Capítulo I, cada Sprint cuenta con un backlog detallado y estos Sprints a su vez cuentan con las tareas a realizar por el equipo de desarrollo, además, cuentan que cuentan con su respectivo caso de prueba para verificar la funcionalidad del sistema.

3.4.1 SPRINT 0

En este Sprint se define un esquema inicial como estructura base del proyecto, diseño del modelo de datos, definición de herramientas y tecnologías con las cuales se desarrolla el prototipo y la implementación del microservicio principal.

Este Sprint se realizó desde el 10 de febrero hasta 25 de febrero del año 2021, en la Tabla 12 se muestra las historias de usuario "Sprint Backlog" con respecto al Sprint 0 y para cada una de sus respectivas tareas a realizar.

<table>
<tr><td colspan="2">Sprint 0 - Backlog</td></tr>
<tr><td>ID</td><td>Nombre de Historia de Usuario</td></tr>
<tr><td>1</td><td>Identificación de Tecnologías</td></tr>
<tr><td colspan="2">Tareas:
- Identificar tecnologías a ser usadas para los microservicios del backend
- Identificar la base de datos ideal para el propósito del sistema
- Identificar las tecnologías a ser usadas para la interfaz de usuario</td></tr>
<tr><td>2</td><td>Diseño de la base de datos</td></tr>
<tr><td colspan="2">Tareas:
- Diseñar el modelo de datos de la base de datos para el servicio principal</td></tr>
<tr><td>3</td><td>Implementación del servicio principal</td></tr>
<tr><td colspan="2">Tareas:
- Definir rutas de la API RESTful
- Definir controladores
- Modulo de registro y autentificación de usuario
- Integración con el microservicio evaluador de programas
- Manejo de roles</td></tr>
</table>

Tabla 12: Sprint 0 Backlog

La Tabla 12 muestra las tecnologías identificadas para la implementación del prototipo en el Sprint 0.

Tecnologías utilizadas en el backend
- NodeJS, para el desarrollo de la API RESTfull - Express Framework - Sequelize, ORM para la interacion entre el servicio principal y la base de datos - PosgreSQL
Tecnologías utilizadas en el frontend
- ReactJS, librería Javascript para construir interfaces de usuario - Material-UI, componentes de ReactJS para un fácil desarrollo web

Tabla 13: Tecnologias usadas para el prototipo del Sprint 0

3.4.2 SPRINT 1

En este Sprint se realiza la implementación del microservicio principal, el servicio evaluador de programas y el diseño prototipo de la interfaz de usuario.

El Sprint fue desarrollado desde el 26 de febrero hasta el 22 de marzo del 2021, en la Tabla 14 se muestra las Historias de Usuario referentes al Sprint 1, así como también, las tareas a realizar.

Tabla 14	
ID	**Nombre de Historia de Usuario**
4	**Implementación de la base de datos**
Tareas: - Configuración del ORM Sequelize para la creación de modelos de base de datos - Implementación de los modelos de la base de datos - Implementación de controladores para la manipulación de los datos de la base de datos	

5	**Implementación del servicio evaluador de programas**
Tareas: - Crear un APIResful que sirva de acceso para el servicio - Evaluación de programas Java - Evaluación de programas C++ - Evaluación de programas Python - Almacenar los casos de prueba	
6	**Diseño inicial de las pantallas de interfaz de usuario**
Tareas: - Investigación de UI/UX para un diseño óptimo - Realización de bocetos para las distintas páginas de la interfaz de usuario	

Tabla 14: Sprint 1 Backlog

3.4.3 SPRINT 2

En este Sprint se realiza la implementación de la interfaz de usuario con los bocetos creados en al anterior sprint, haciendo uso de las tecnologías de desarrollo frontend: ReactJS y Material UI, al culminar la implementación del frontend y ya culminado el desarrollo de los microservicios se procede a integrar estos, haciendo uso de sus API's RESTful, una vez culminado el proceso de integración se procede al despliegue del sistema en un servidor.

El Sprint fue desarrollado desde el 22 de marzo hasta el 28 de abril del 2021, en la Tabla 15 se muestra las Historias de Usuario referentes al Sprint 2, así como también, las tareas a realizar.

Tabla 15	
Sprint 2 - Backlog	
ID	**Nombre de Historia de Usuario**
7	**Implementación de la interfaz de usuario**

Tareas: - Implementación del formulario de registro y formulario de acceso a la plataforma - Implementación del módulo de listado, creación, edición y envió para evaluación de los Problemas de Algoritmia - Implementación de la sección de Concursos de Programación - Implementación de la sección del estado de los Problemas - Implementación de la pantalla principal o de bienvenida	
8	**Integración de los microservicios**
Tareas: - Integrar el microservicio de interfaz de usuario con el servicio principal haciendo uso del protocolo de comunicación HTTP/HTTPS - Integrar el microservicio de interfaz de usuario con el servicio evaluador de programas haciendo uso del protocolo de comunicación HTTP/HTTPS - Integrar el microservicio principal con el microservicio evaluador de programas para hacer la evaluación de los programas con sus respectivos casos de prueba	
9	**Despliegue del sistema basado en microservicios**
Tareas: - Investigar que plataforma de host de servicios se adapta eficientemente a la arquitectura de nuestro sistema - Desplegar el sistema basado en microservicios en la plataforma elegida para realizar las pruebas correspondientes	

Tabla 15: Sprint 2 Backlog

Las imágenes mostradas a continuación corresponden a el incremento de este sprint.

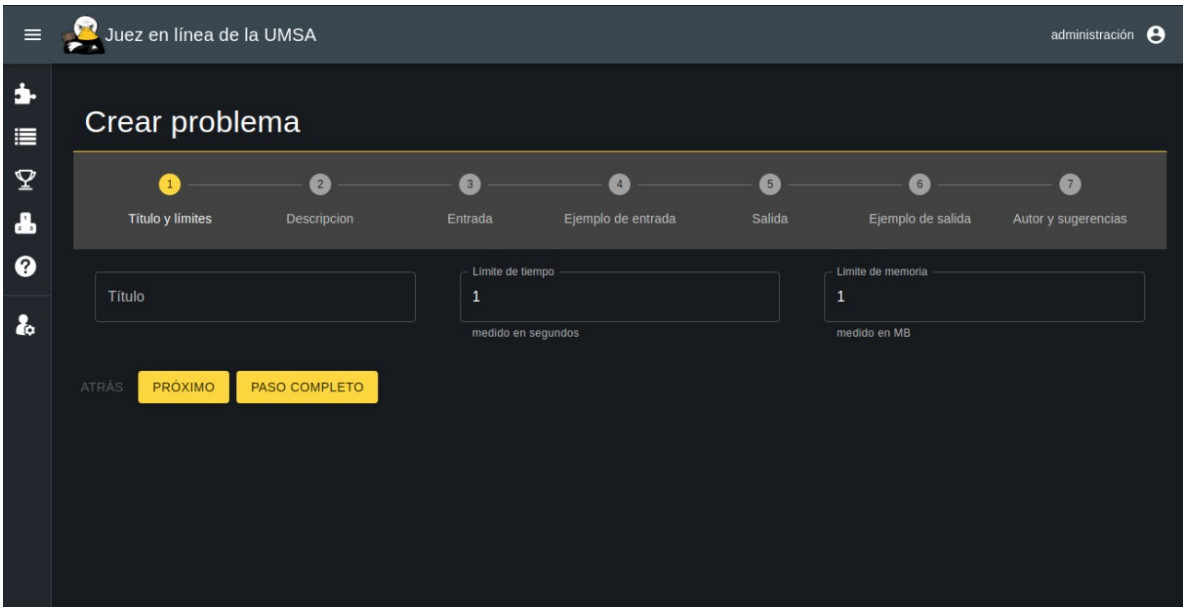


Figura 9: Creación de Problemas

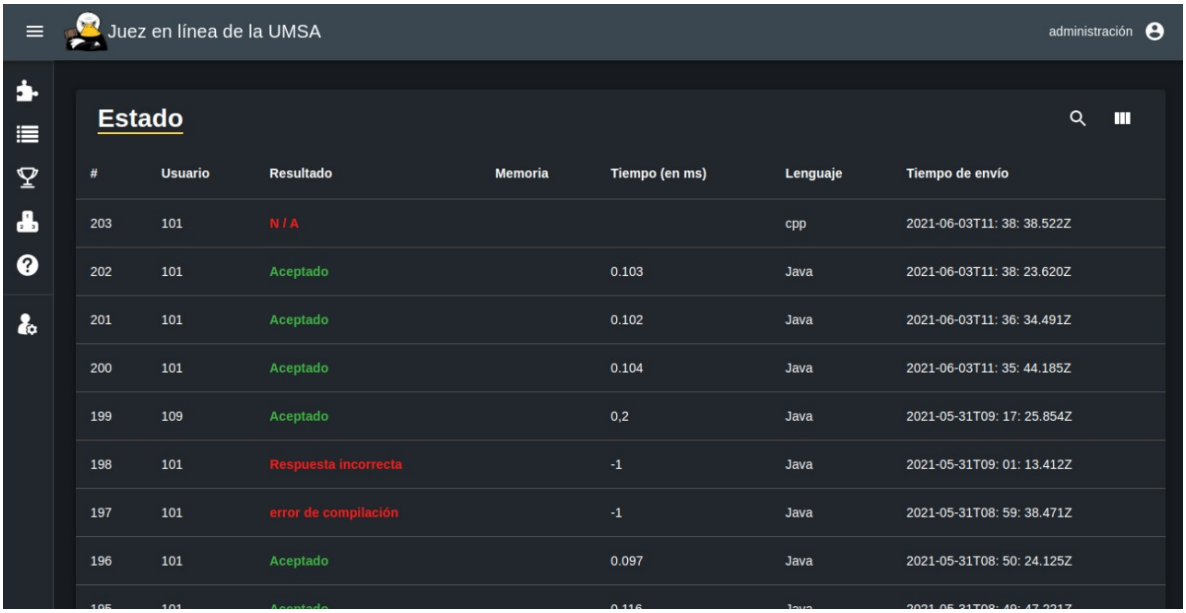


Figura 10: Página del Estado de las Soluciones

Figura 11: Página de Inicio

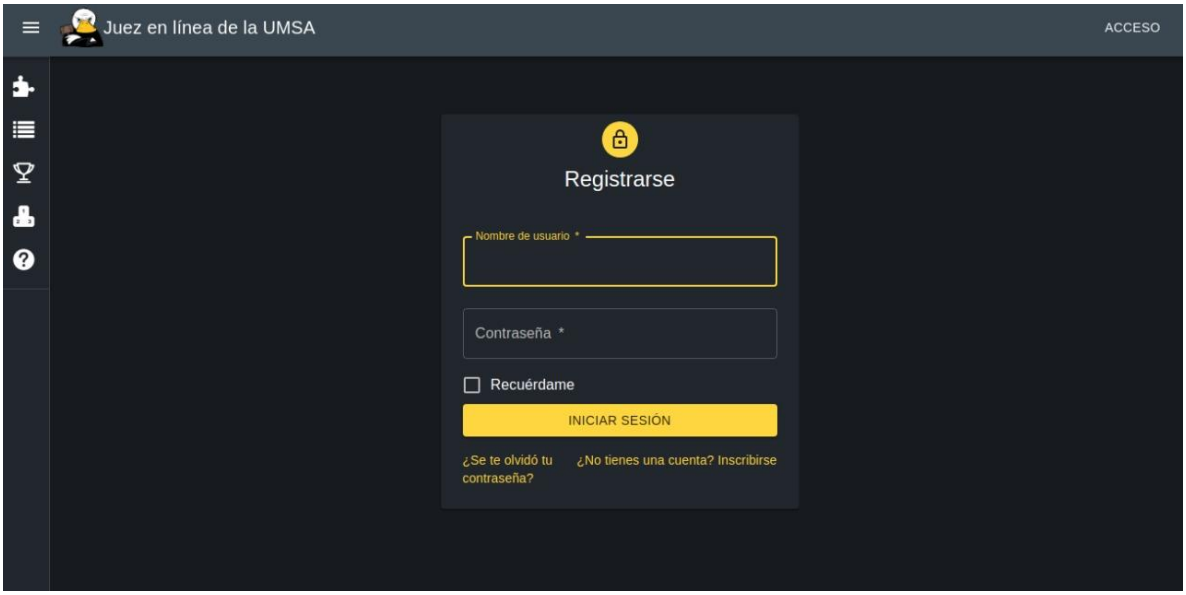


Figura 12: Página de Ingreso

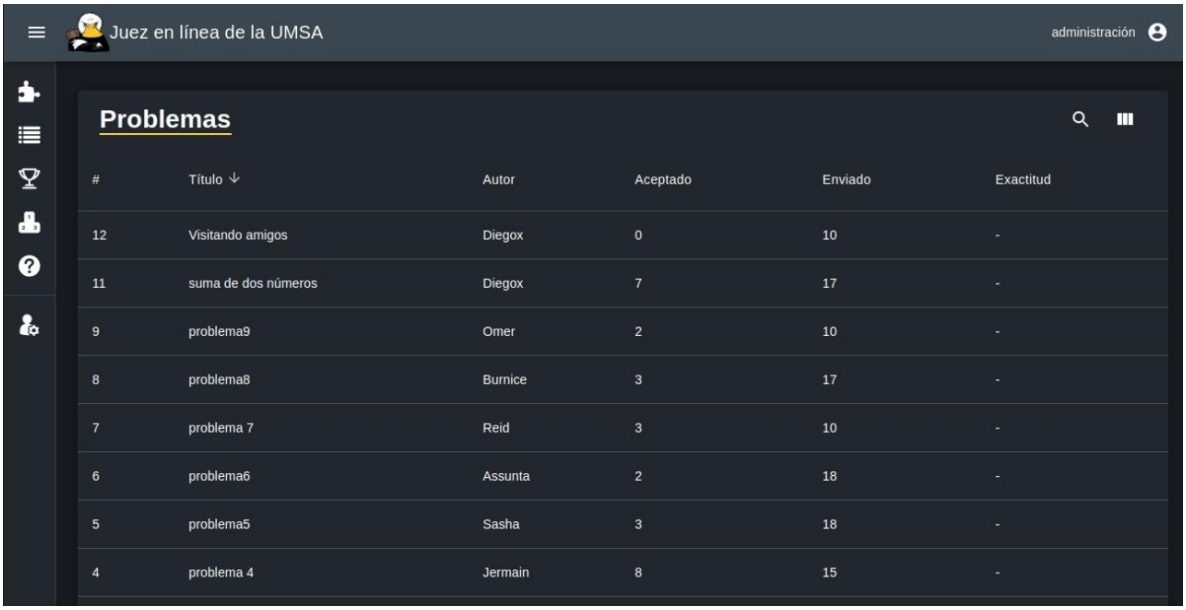


Figura 13: Página del Repositorio de Problemas

Figura 14: Ejemplo de Problema

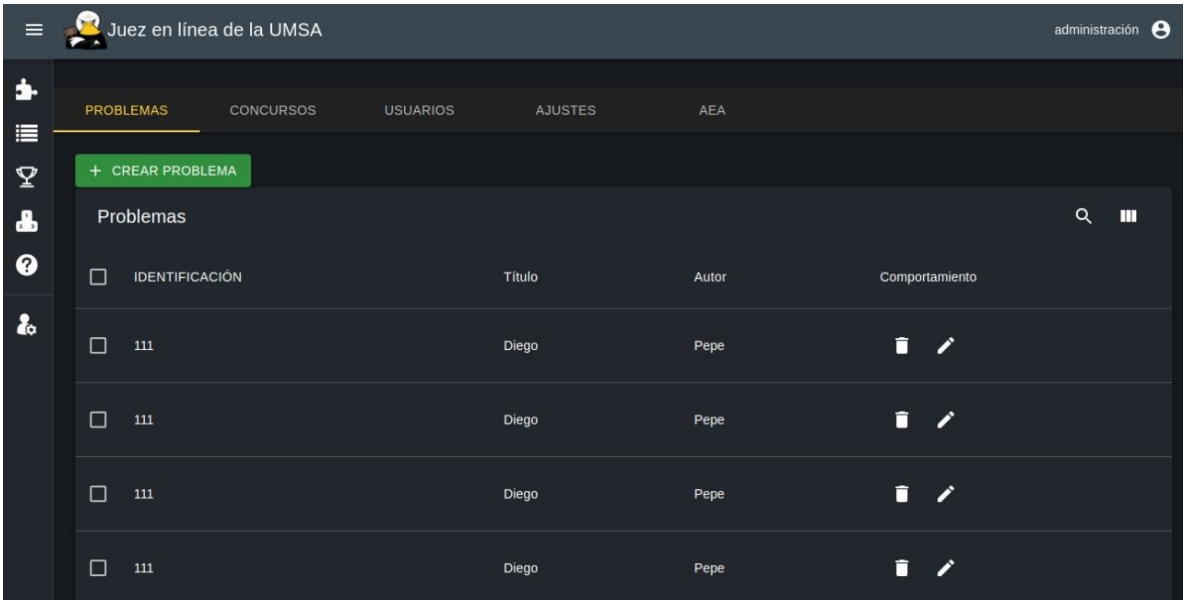


Figura 15: Página de Administración

3.5 MODELO DE BASE DE DATOS

Figura 16. Modelo de Base de Datos

Fuente: Elaboración Propia

3.6 ARQUITECTURA DEL SISTEMA BASADO EN MICROSERVICIOS

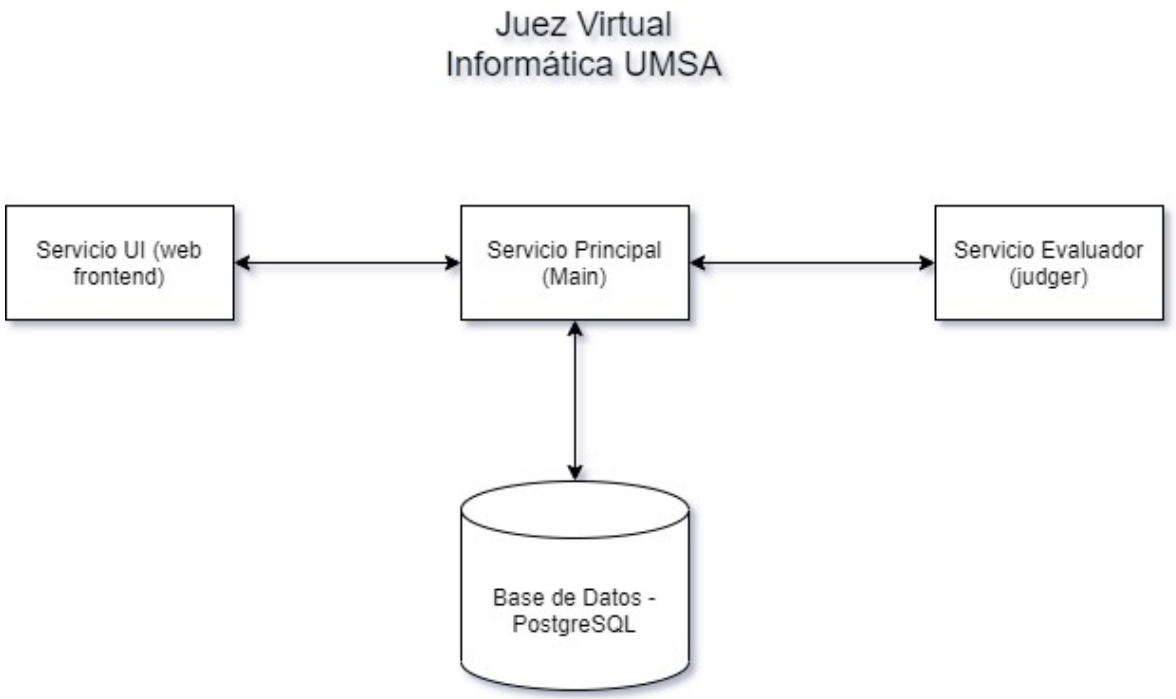


Figura 17 Arquitectura del juez virtual

Compuesta inicialmente por tres servicios y una base de datos, los cuales son:

- *Main service* (Servicio Principal): este servicio se encarga del manejo de usuarios, el registro de problemas, *contests* (competencias de programacion), *submissions* (envio de soluciones), estado y todo lo que tenga que ver a los reportes. Este servicio hace uso de una base de datos SQL (Postgres)
- Judger service (o servicio evaluador): este servicio es el encargado de evaluar (juzgar) los problemas y de almacenar los casos de prueba y las soluciones de los problemas, recibe una serie de parámetros para cumplir con este propósito tales como: la información del problema, la ubicación donde los casos de prueba estan almacenados etc.
- UI service (web frontend): este servicio se encarga de mostrar la interfaz gráfica del juez y se comunica con los dos servicios mencionados anteriormente.

3.7 MODELO ENTIDAD-RELACIÓN

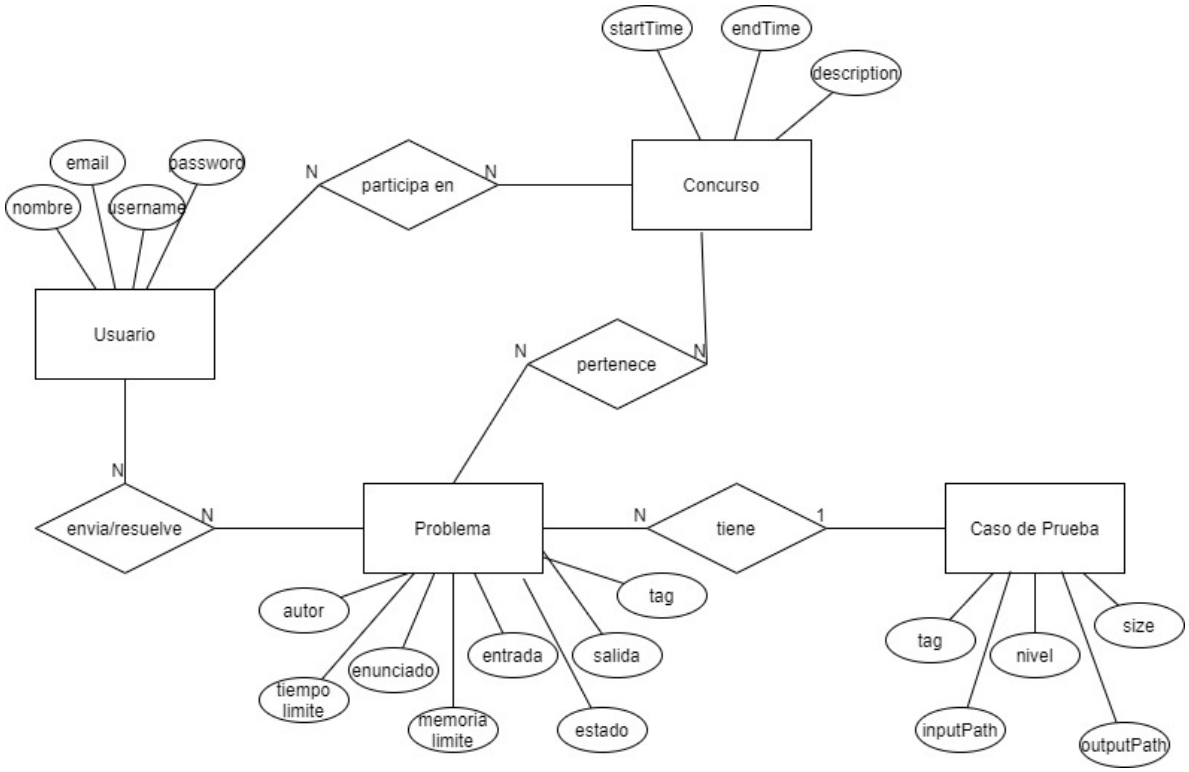


Figura 18: Modelo Entidad-Relación

CAPITULO IV

EVALUACIÓN DE RESULTADOS

4.1 INTRODUCCIÓN

Una vez desarrollado el prototipo se realiza la fase experimental de la investigación el cual consiste en desarrollar un plan o estrategia para obtener información para alcanzar el objetivo de estudio y para contestar la interrogante de conocimiento que se ha planteado y principalmente para analizar la certeza de la formulación de la hipótesis de la investigación. Se realiza la experimentación determinando las siguientes actividades:

- Determinar la población.
- Determinar la muestra.
- Proceso de experimentación.
- Análisis de los resultados obtenidos.

Considerando la hipótesis planteada en la sección 1.4:

4.2 PRUEBAS DE CARGA

Se ha realizado una serie de pruebas de carga para ver cómo responde el sistema de microservicios ante la carga de datos y ver la resiliencia del mismo.

Esto no ayudara a identificar:

- La máxima capacidad de operación del sistema
- Determinar si la infraestructura actual es suficiente para que el sistema cumpla su objetivo
- Sustentabilidad del sistema con respecto a la carga máxima
- Número de peticiones concurrentes que el sistema puede soportar y escalabilidad para aceptar más peticiones

Para las pruebas se enviarán inicialmente 5 peticiones concurrentes al sistema, luego, se enviarán 5 peticiones más dando un total de 10 peticiones concurrentes para la segunda iteración, y así sucesivamente hasta llegar a la carga máxima que el sistema puede soportar. Las peticiones serán enviadas en intervalos de 10 segundos.

Los resultados se detallan en la siguiente tabla.

Prueba de Carga			
Número de Iteración	**Número de peticiones concurrentes**	**Tiempo de Respuesta Promedio (en milisegundos)**	**Observación**
1	5	1065	Buen desempeño
2	10	1087	Buen desempeño
3	15	1137	Buen desempeño
4	20	1284	Buen desempeño
5	25	1576	Buen desempeño
6	30	1663	El tiempo de respuesta se va incrementando
7	35	1817	El tiempo de respuesta se va incrementando
8	40	1917	El tiempo de respuesta se va incrementando
9	45	2103	El tiempo de respuesta de incremento en un segundo con respecto a la iteración numero 1

10	50	2139	El tiempo de respuesta sobrepasa los 2 segundos

Tabla 16: Pruebas de Carga

De la Tabla 16, se puede observar que el número óptimo de peticiones concurrentes con un tiempo de respuesta aceptable y amigable para el usuario es de 50.

4.3 PLANTEAMIENTO DE LA HIPÓTESIS

Considerando la hipótesis que se plante en el Capítulo I: "La implementación de microservicios para la evaluación de programas basados en casos de prueba **mejora** el rendimiento del juez virtual de la Carrera de Informática con una confiabilidad de **95%.**"
De esta hipótesis planteada se identificó la hipótesis nula la cual indica que no hay diferencia, es decir, el rendimiento del juez actual es igual al del juez basado en microservicios y la hipótesis alternativa la cual indica que existe una diferencia significativa, es decir, el rendimiento del juez basado en microservicios es mayor (el tiempo de respuesta es menor):

H_0: La implementación de microservicios para la evaluación de programas basados en casos de prueba **no mejora** el rendimiento del juez virtual de la Carrera de Informática con una confiabilidad de **95%.**

H_1: La implementación de microservicios para la evaluación de programas basados en casos de prueba **mejora** el rendimiento del juez virtual de la Carrera de Informática con una confiabilidad de **95%.**

4.3.1 DEMONSTRACIÓN DE LA HIPÓTESIS

Para evaluar a variable dependiente se utilizará la prueba T-Student, que es una distribución muestral o poblacional de la diferencia de medias. Esta distribución se identifica por los grados de libertad que constituye el número de valores elegidos libremente. Son determinantes, ya que indican que valor debemos esperar, dependiendo del tamaño de los grupos que se comparan.

4.3.1.1 SUJETO DE ESTUDIO

Los sujetos de estudio de la investigación, corresponden a peticiones que se envían a los jueces virtuales. Se toman en cuenta dos grupos:

- El primer grupo de peticiones que se realizaron a el nuevo sistema basado en arquitectura de microservicios (grupo experimental)
- El segundo grupo de peticiones que se enviaron a el juez virtual con el que la carreara de Informática cuenta actualmente.

4.3.1.2 TAMAÑO DE LA MUESTRA

La población considerada para esta investigación consta de una serie lotes, cada lote consta de 50 peticiones y se enviaran 100 lotes.

Para obtener una muestra probabilística representativa, asumiendo que la población objetivo es grande, pasamos a determinar las siguientes relaciones:

Tamaño provisional de la muestra:

$$n = \frac{s^2}{v^2} = \frac{varianza\ de\ la\ muestra}{varianza\ de\ la\ poblacion} \qquad (1)$$

Tamaño óptimo de la muestra:

$$n = \frac{n'}{1 + \frac{n'}{N}} \qquad (2)$$

Donde:

- N: tamaño de la población
- v^2: varianza de la población al cuadrado
- s^2: desviación estándar o error estándar: 0.05

Se halla la varianza de la muestra en términos de la probabilidad y la varianza de la población en función a la desviación estándar, se obtiene:

$$s^2 = p(1-p) = 0.095(1-0.095) = 0.0475$$

$$v^2 = \sigma^2 = 0.05^2 = 0.0025$$

Remplazando en la ecuación (1) se tiene:

$$n = \frac{s^2}{v^2} = \frac{0.0475}{0.0025} = 19$$

Donde $n = 19$ será el tamaño provisional de la muestra

Ahora n' se reemplaza en la relación (2) y se obtiene:

$$n = \frac{n'}{1 + \frac{n'}{N}} = \frac{19}{1 + \frac{19}{100}} = 17$$

Por tanto, el tamaño de la muestra es de n = 17, que se aplica en los lotes de peticiones que se envían al juez virtual, este tamaño de muestra se aplica a los dos grupos de lotes de peticiones descritos anteriormente.

4.3.1.3 DESCRIPCIÓN DEL PROCESO

Las pruebas se enviaron en lotes de peticiones individuales, tanto al juez virtual actual, así como también, al juez virtual basado en microservicios.

Los resultados obtenidos se muestran en la siguiente tabla:

Grupo experimental	
Nro. de lote	**Tiempo de respuesta (en segundos)**
1	4.2
2	3.1
3	3.4
4	3.2
5	4.8
6	3.6
7	4.4
8	5.1
9	4.2
10	3.6
11	3.5
12	5.2
13	2.9
14	3.3
15	3.9
16	2
17	3.1
TOTAL:	63.5
PROMEDIO:	3.735
Nota: Envió de lotes de peticiones enviados al juez con una arquitectura monolítica.	

Tabla 17: Peticiones por lotes al juez con arquitectura monolítica

Grupo experimental	
Nro. de lote	**Tiempo de respuesta (en segundos)**
1	2.6
2	2.3
3	3.1
4	3.4
5	2.3
6	3.9
7	4.6
8	2.2
9	3.1
10	2.6
11	3.0
12	2.2
13	2.2
14	2.1
15	3.4
16	2.2
17	2
TOTAL:	47.2
PROMEDIO:	2.776

Nota: Envió de lotes de peticiones enviados al juez con una arquitectura de microservicios.

Tabla 18: Peticiones por lotes al juez con arquitectura de microservicios

De las muestras pequeñas, la formula T-Student esta expresada de la siguiente forma:

$$t = \frac{\bar{x}_1 - \bar{x}_2}{\sqrt{\frac{(n_1 - 1)s_1^2 + (n_2 - 1)s_2^2}{(n_1 + n_2) - 2}\left(\frac{1}{n_1} + \frac{1}{n_2}\right)}} \qquad (3)$$

Donde:

- $\bar{x}_1$: es la medida del grupo experimental
- $\bar{x}_2$: es la medida del grupo de control
- s_1^2: es la desviación estándar del grupo experimenta
- s_2^2: es la desviación estándar del grupo de control
- n_1: el tamaño del grupo de experimental
- n_2: el tamaño del grupo de control

Los grados de libertad están dados por la siguiente formula:

$$gl = (n_1 + n_2) - 2 \qquad (4)$$

De las tablas 16 y 17 se obtienen las siguientes medias:

$$\bar{x}_1 = 3.735$$

$$\bar{x}_2 = 2.776$$

La desviación estándar está dada por:

$$s_1^2 = \sum_{i=1}^{n} \frac{(x_i - \bar{x}_1)^2}{n - 1} \qquad (5)$$

Reemplazando datos:

$$s_1^2 = \sum_{i=1}^{n} \frac{(x_i - \bar{x}_1)^2}{n - 1} = \frac{11.238}{16} = 0.7024$$

$$s_2^2 = \sum_{i=1}^{n} \frac{(x_i - \bar{x}_2)^2}{n-1} = \frac{8.530}{16} = 0.5330$$

Finalmente reemplazando en la ecuación (3), se tiene:

$$t = \frac{3.735 - 2.776}{\sqrt{\frac{(17-1)0.702 + (17-1)0.533}{(17+17)-2}\left(\frac{1}{17} + \frac{1}{17}\right)}} = \frac{0.959}{0.269} = 3.55$$

Para hallar los grados de libertad reemplazamos en la ecuación (4):

$$gl = (n_1 + n_2) - 2 = 32$$

Una vez obtenidos todos los resultados utilizando la tabla T-Student, se busca el valor en el cual se compara tomando 32 como grados de libertad, los niveles de confianza elegido (0.05) adquieren el significado de 0,05 significa que los grupos difieren entre un 95%, obteniendo un error de posibilidad de error:

G	**Nivel de confianza 0.05**
32	1.6939
Nota: Nivel de confianza	

Tabla 19: Nivel de Confianza

4.4 ANÁLISIS DE RESULTADOS

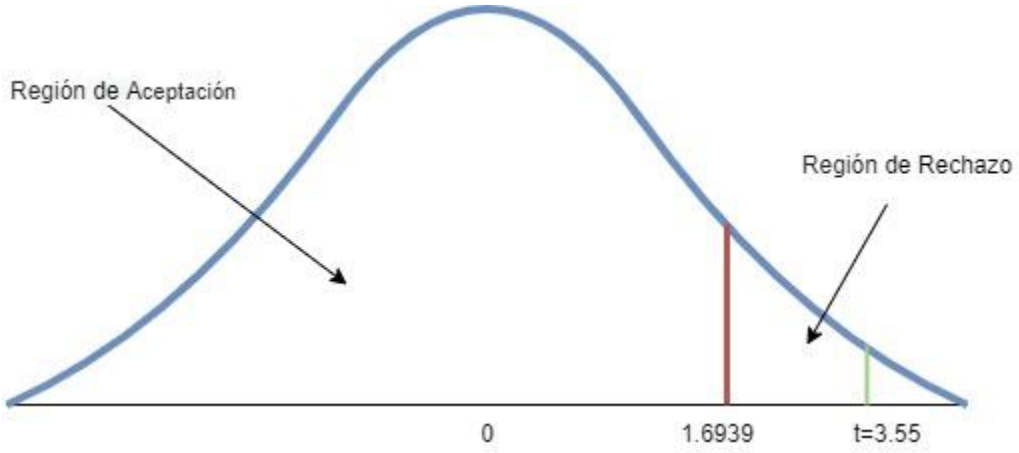


Figura 19: Región de Aceptación

En la figura 18 se muestra la región de aceptación y rechazo en función al valor critico 1.6939 (véase el Anexo C).

De la hipótesis nula y la hipótesis alternativa se tiene:

H_0: $\bar{x}_1 = \bar{x}_2$

H_1: $\bar{x}_1 > \bar{x}_2$

Entonces, se rechaza H_0 y se acepta H_1.

Nombre de la variable	**Indicador**	**Resultado**	**Definición operacional**
La mejora de rendimiento según la carga de datos	T-Student	t =3.55 con niveles de confianza 0.05 (3.55>1,6939)	Se redujo el tiempo de respuesta de las peticiones haciendo uso de la arquitectura de microservicios.
Nota: Resultados finales de la variable dependiente			

Tabla 20: Tabla de Análisis de Resultados

4.5 CONCLUSIONES

Dada la hipótesis:

“La implementación de microservicios para la evaluación de programas basados en casos de prueba **mejora** el rendimiento del juez virtual de la Carrera de Informática con una confiabilidad de **95%**”

Con los resultados expuestos en la tabla 19 podemos concluir que: El valor de 0.05 ($3{,}55 > 1{,}6939$) significa que existe un 95% de confiabilidad con un error del 5%.

Los resultados alcanzados, ayudan a comprobar la hipótesis planteada.

CAPITULO V

CONCLUSIONES Y RECOMENDACIONES

5.1 CONCLUSIONES

La arquitectura de microservicios es una forma de diseñar y construir aplicaciones altamente escalables, mantenibles y muy eficientes de manera ágil. Haciendo uso de esta arquitectura se ha construido un evaluador de casos de prueba (juez virtual) que responde de manera eficiente a las peticiones de los usuarios y se ha logrado cumplir con los objetivos planteados.

De acuerdo a los resultados obtenidos se puede concluir:

- Se diseño e implementó el sistema basado en la arquitectura de microservicios como se indica en el Capítulo 3.
- Con respecto a la hipótesis planteada en el capítulo primero, se llegó a demostrar que el sistema basado en la arquitectura de microservicios para la evaluación de programas, es aceptado en un 95%, permitiendo atender las peticiones de los usuarios de una forma más eficiente.

En cuanto a los objetivos específicos alcanzados se concluye que:

- El sistema para la evaluación de programas basados en casos de prueba evalúa de manera automática, rápida, segura y confiable programas basados en casos de prueba.
- El sistema basado en la arquitectura de microservicios mejora el tiempo de respuesta del resultado de la solución y da veredicto pertinente.
- El sistema basado en la arquitectura de microservicios crea un repositorio catalogado de problemas creados por la comunidad de la Carrera de Informática en la que los estudiantes podrán practicar y mejorar sus habilidades en programación.

5.2 RECOMENDACIONES

Si bien la arquitectura de servicios nos ayuda a solucionar varios problemas que representa mantener un sistema monolítico, debemos tomar en cuenta que la arquitectura de microservicios también conlleva superar algunos desafíos al momento de su implementación, tal como se vio en secciones anteriores.

En cuanto a la experiencia de usuario y características adicionales se recomienda:

- Agregar un servicio el cual pueda ayudar a detectar la copia entre soluciones enviadas por los usuarios, para evitar así la obtención de puntos no merecidos y no dañar el proceso de aprendizaje al momento de practicar algoritmia.
- Hacer uso de herramientas DevOps tales como Kubernetes, Openshift, etc, para mejorar la escalabilidad y mantenibilidad del sistema.
- Integración con otras plataformas de aprendizaje de algoritmos y programación.

REFERENCIAS

Beck, K., Beedle, M., Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., . . . Sutherland, J. &. (2001). *Principios de Manifesto Agil*. Obtenido de http://agilemanifesto.org/iso/es/principles.html

Cuba, J. G. (2015). Tutor Inteligente para el Proceso de Aprendizaje en Algoritmos y Programacion en el Lenguaje Java.

Group, P. D. (2020). *Postgresql Developer Group*. Obtenido de https://www.postgresql.org/developer/

GuiaDev. (2020). *GuiaDev - Introduccion a Docker*. Obtenido de https://guiadev.com/introduccion-a-docker-4-microservicios/

Jabón, M. y. (2013). Juez Automático para la Evaluación de Problemas de Programación en los Primeros Cursos de Estudios de Informática.

R, C. (2015). Diseño e Implementación de un Juez En Línea Para El Desarrollo De Competencias Algorítmicas En La Universidad Libre.

ReactJS. (2020). *React - Getting started*. Obtenido de https://reactjs.org/docs/getting-started.html

RedHat. (2021). *RedHat - Microservicios*. Obtenido de https://www.redhat.com/es/topics/microservices/what-are-microservices

Schwaber, K. &. (2016). *Scrum Guides*. Obtenido de http://scrumguides.org/docs/scrumguide/v1/Scrum-Guide-US.pdf

Wikipedia. (2020). *Wikipedia - Ingenieria de Software*. Obtenido de https://es.wikipedia.org/wiki/Ingenier%C3%ADa_de_software

WIkipedia. (2020). *Wikipedia - Metodologia de desarrollo de Software*. Obtenido de https://es.wikipedia.org/wiki/Metodolog%C3%ADa_de_desarrollo_de_software

Wikipedia. (2020). *WIkipedia, Docker*. Obtenido de WIkipedia, Docker: https://es.wikipedia.org/wiki/Docker_(software)

ANEXOS

ANEXO A - IMPLEMENTACIÓN DE MODELO DE BASE DE DATOS

```
'use strict';

module.exports = (sequelize, DataTypes) => {
  const Problem = sequelize.define('problem', {
    title: {
      type: DataTypes.STRING,
      allowNull: false,
      trim: true
    },
    author: {
      type: DataTypes.STRING,
      trim: true
    },
    statement: {
      type: DataTypes.TEXT,
      allowNull: false,
    },
    inputDescription: {
      type: DataTypes.TEXT,
      allowNull: false
    },
    inputExample: {
      type: DataTypes.TEXT,
      allowNull: false
    },
    outputDescription: {
      type: DataTypes.TEXT,
      allowNull: false
    },
    outputExample: {
      type: DataTypes.TEXT,
      allowNull: false
    },
    timeLimit: {
      type: DataTypes.INTEGER,
      allowNull: false
    },
    memoryLimit: {
      type: DataTypes.INTEGER,
      allowNull: false
    },
    hints: {
```

```
43.       type: DataTypes.TEXT,
44.     }
45.   }, {});
46.   Problem.associate = function (models) {
47.     Problem.belongsTo(models.user, {
48.       foreignKey: 'userId'
49.     });
50.     Problem.hasMany(models.testCase);
51.     Problem.hasMany(models.submission);
52.   };
53.
54.   return Problem;
55. };
56.
```

ANEXO B - IMPLEMENTACIÓN DE EVALUADOR DE PROGRAMAS PARA EL LENGUAJE DE PROGRAMACIÓN JAVA

```
1. const { spawn, exec, execSync } = require("child_process");
2. const Runner = require("./Runner");
3. const fileManager = require('../utils/fileManager');
4. const path = require('path');
5. const appRoot = require('app-root-path');
6. const toKebabCase = require('./../utils/toKebabCase');
7. const fs = require('fs');
8.
9. const generateSuccessCompilationLog = (stderr = null, stdout = null,
   exitCode = null, killed = null) => {
10.   const result = 'Compiled succesfully';
11.   return {
12.     stderr,
13.     stdout,
14.     exitCode,
15.     killed,
16.     result
17.   };
18. }
19.
20. const generateCompilationErrorLog = (error) => {
21.   const result = 'Compilation error';
22.   let message = String(error.message || '');
23.   message = message.substring(message.indexOf('error: '));
24.   return {
25.     code: error.code,
26.     killed: error.killed,
27.     message,
28.     signal: error.signal,
29.     result
30.   };
31. }
32.
33. const compareOutput = (folderName, caseNo, id) => {
34.   const originalTestCasesPath = `${appRoot}/test-
   cases/${folderName}/output/output${caseNo}.txt`;
35.   const solutionTestCasesPath = `${appRoot}/${id}/judge-
   output/output${caseNo}.txt`;
36.   try {
37.     const original = fs.readFileSync(originalTestCasesPath, 'utf-8');
```

```
38.     const solution = fs.readFileSync(solutionTestCasesPath, 'utf-8');
39.     return original === solution;
40.   } catch (error) {
41.     console.log('error: ', error);
42.   }
43. }
44. class JavaRunner extends Runner {
45.
46.   constructor(problem, language, solution, testCases) {
47.     super(problem, language, solution, testCases);
48.   }
49.
50.   async prepareFiles(solution, id) {
51.     const sourceDirectory = path.resolve(`${appRoot}`, `solutions-folder`,
    solution);
52.     const destinationDirectory = path.resolve(`${appRoot}`, `${id}`, `run-
    folder`, 'Main.java');
53.     await fileManager.createDirectory(path.resolve(`${appRoot}`, `${id}`,
    'judge-output'));
54.     await fileManager.copyFile(sourceDirectory, destinationDirectory);
55.   }
56.   // compile java source file
57.
58.
59.   compile(id) {
60.     return new Promise((resolve, reject) => {
61.       const options = {};
62.       const cmd = `javac ${appRoot}/${id}/run-folder/Main.java`;
63.
64.       const response = {
65.         success: true,
66.         compilationError: false,
67.         veredict: '',
68.         time: -1,
69.         displayVeredict: '',
70.         totalTestCases: 0,
71.         judgedTestCases: 0,
72.         judgments: [],
73.         failedJudgment: {},
74.       }
75.
76.       // return {
77.       //   code: error.code,
78.       //   killed: error.killed,
79.       //   message,
```

```
80.       //   signal: error.signal,
81.       //   result
82.       // };
83.
84.       const javac = exec(cmd, options, (error, stdout, stderr) => {
85.         let compilationResult;
86.         if (error) {
87.           compilationResult = generateCompilationErrorLog(error);
88.           response.compilationError = true;
89.           response.displayVeredict = 'Compilation Error';
90.           response.veredict = 'COMPILATION_ERROR';
91.           response.success = false;
92.           response.message = compilationResult.message;
93.           reject(response);
94.         }
95.         resolve(generateSuccessCompilationLog(stderr, stdout,
    javac.exitCode, javac.killed));
96.       });
97.     });
98.   }
99.
100.        execute(cmd, options) {
101.          return new Promise((resolve, reject) => {
102.            const process = exec(cmd, options, (error, stdout, stderr) =>
    {
103.              if (error) {
104.                reject(error);
105.              }
106.              const result = {
107.                stdout,
108.                stderr,
109.                exitCode: process.exitCode,
110.              }
111.              resolve(result);
112.            });
113.          });
114.        }
115.
116.        async run(testCases, problem, id) {
117.          const folderName = toKebabCase(testCases);
118.          const timeLimit = problem.timeLimit;
119.          const memoryLimit = problem.memoryLimit;
120.          const inputTestCasesFolder = `${appRoot}/test-
    cases/${folderName}/input`;
121.
```

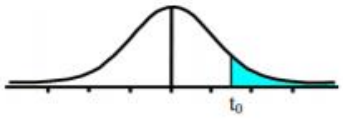
```
122.          const options = {
123.            encoding: 'utf8',
124.            shell: '/bin/bash',
125.            timeout: timeLimit * 1000,
126.            killSignal: 'SIGKILL',
127.            env: null
128.          };
129.
130.          const testCaseFiles = fs.readdirSync(inputTestCasesFolder, {});
131.          const response = {
132.            success: true,
133.            compilationError: false,
134.            veredict: '',
135.            time: -1,
136.            displayVeredict: '',
137.            totalTestCases: testCaseFiles.length,
138.            judgedTestCases: 0,
139.            judgments: [],
140.            failedJudgment: {},
141.          }
142.
143.          let caseNo = 1;
144.          for (const fileName of testCaseFiles) {
145.            let judgment = {};
146.            const cmd = `java -cp ${appRoot}/${id}/run-folder Main <
    ${inputTestCasesFolder}/${fileName} > ${appRoot}/${id}/judge-
    output/output${caseNo}.txt`;
147.            const startTime = Date.now();
148.            try {
149.              // success?
150.              await this.execute(cmd, options);
151.
152.              const endTime = Date.now();
153.              const totalTime = (endTime - startTime) / 1000.0;
154.              judgment.time = totalTime;
155.
156.              const accepted = compareOutput(folderName, caseNo, id);
157.              if (accepted) {
158.                judgment.veredict = 'ACCEPTED';
159.                judgment.displayVeredict = 'Accepted';
160.                response.judgments.push(judgment);
161.              } else {
162.                judgment.veredict = 'WRONG_ANSWER';
163.                judgment.displayVeredict = 'Wrong Answer';
164.                response.success = false;
```

```
165.                 response.veredict = judgment.veredict;
166.                 response.displayVeredict = judgment.displayVeredict;
167.                 response.failedJudgment = judgment;
168.                 response.judgedTestCases = caseNo;
169.                 break;
170.               }
171.             } catch (error) {
172.               // error
173.               const endTime = Date.now();
174.               const totalTime = (endTime - startTime) / 1000.0;
175.               judgment.time = totalTime;
176.
177.               if (error.killed && error.signal === 'SIGKILL' && error.code
    === null) {
178.                 judgment.exitCode = 256;
179.                 judgment.veredict = judgment.displayVeredict = 'TLE';
180.               } else if (judgment.exitCode === 136) {
181.                 judgment.veredict = 'SIGFPE';
182.                 judgment.displayVeredict = 'RTE(SIGFPE)';
183.               } else if (judgment.exitCode === 139) {
184.                 judgment.veredict = 'SIGSEGV';
185.                 judgment.displayVeredict = 'RTE(SIGSEGV)';
186.               } else if (judgment.exitCode === 153) {
187.                 judgment.veredict = 'SIGXFSZ';
188.                 judgment.displayVeredict = 'RTE(SIGXFSZ)';
189.               } else if (judgment.exitCode == 134) {
190.                 judgment.veredict = 'SIGABRT';
191.                 judgment.displayVeredict = 'RTE(SIGABRT)';
192.               } else {
193.                 judgment.veredict = 'NZEC';
194.                 judgment.displayVeredict = 'RTE(NZEC)';
195.               }
196.               response.success = false;
197.               response.veredict = judgment.veredict;
198.               response.displayVeredict = judgment.displayVeredict;
199.               response.failedJudgment = judgment;
200.               response.judgedTestCases = caseNo;
201.               break;
202.             }
203.             caseNo = caseNo + 1;
204.           }
205.           if (response.success) {
206.             response.veredict = 'ACCEPTED';
207.             response.displayVeredict = 'Accepted';
```

```
208.             response.time = response.judgments.map(judgment =>
    judgment.time).reduce((prev, current) => Math.max(prev, current), -1);
209.             response.judgedTestCases = caseNo - 1;
210.           }
211.           return response;
212.         }
213.
214.         async clean(id) {
215.           await fileManager.delete(path.resolve(`${appRoot}`, `${id}`))
216.         }
217.
218.       }
219.
220.       module.exports = JavaRunner;
221.
```

ANEXO C – TABLA T-STUDENT

Tabla t-Student



Grados de libertad	0.25	0.1	0.05	0.025	0.01	0.005
1	1.0000	3.0777	6.3137	12.7062	31.8210	63.6559
2	0.8165	1.8856	2.9200	4.3027	6.9645	9.9250
3	0.7649	1.6377	2.3534	3.1824	4.5407	5.8408
4	0.7407	1.5332	2.1318	2.7765	3.7469	4.6041
5	0.7267	1.4759	2.0150	2.5706	3.3649	4.0321
6	0.7176	1.4398	1.9432	2.4469	3.1427	3.7074
7	0.7111	1.4149	1.8946	2.3646	2.9979	3.4995
8	0.7064	1.3968	1.8595	2.3060	2.8965	3.3554
9	0.7027	1.3830	1.8331	2.2622	2.8214	3.2498
10	0.6998	1.3722	1.8125	2.2281	2.7638	3.1693
11	0.6974	1.3634	1.7959	2.2010	2.7181	3.1058
12	0.6955	1.3562	1.7823	2.1788	2.6810	3.0545
13	0.6938	1.3502	1.7709	2.1604	2.6503	3.0123
14	0.6924	1.3450	1.7613	2.1448	2.6245	2.9768
15	0.6912	1.3406	1.7531	2.1315	2.6025	2.9467
16	0.6901	1.3368	1.7459	2.1199	2.5835	2.9208
17	0.6892	1.3334	1.7396	2.1098	2.5669	2.8982
18	0.6884	1.3304	1.7341	2.1009	2.5524	2.8784
19	0.6876	1.3277	1.7291	2.0930	2.5395	2.8609
20	0.6870	1.3253	1.7247	2.0860	2.5280	2.8453
21	0.6864	1.3232	1.7207	2.0796	2.5176	2.8314
22	0.6858	1.3212	1.7171	2.0739	2.5083	2.8188
23	0.6853	1.3195	1.7139	2.0687	2.4999	2.8073
24	0.6848	1.3178	1.7109	2.0639	2.4922	2.7970
25	0.6844	1.3163	1.7081	2.0595	2.4851	2.7874
26	0.6840	1.3150	1.7056	2.0555	2.4786	2.7787
27	0.6837	1.3137	1.7033	2.0518	2.4727	2.7707
28	0.6834	1.3125	1.7011	2.0484	2.4671	2.7633
29	0.6830	1.3114	1.6991	2.0452	2.4620	2.7564
30	0.6828	1.3104	1.6973	2.0423	2.4573	2.7500
31	0.6825	1.3095	1.6955	2.0395	2.4528	2.7440
32	0.6822	1.3086	1.6939	2.0369	2.4487	2.7385
33	0.6820	1.3077	1.6924	2.0345	2.4448	2.7333
34	0.6818	1.3070	1.6909	2.0322	2.4411	2.7284
35	0.6816	1.3062	1.6896	2.0301	2.4377	2.7238
36	0.6814	1.3055	1.6883	2.0281	2.4345	2.7195
37	0.6812	1.3049	1.6871	2.0262	2.4314	2.7154
38	0.6810	1.3042	1.6860	2.0244	2.4286	2.7116
39	0.6808	1.3036	1.6849	2.0227	2.4258	2.7079
40	0.6807	1.3031	1.6839	2.0211	2.4233	2.7045
41	0.6805	1.3025	1.6829	2.0195	2.4208	2.7012
42	0.6804	1.3020	1.6820	2.0181	2.4185	2.6981
43	0.6802	1.3016	1.6811	2.0167	2.4163	2.6951
44	0.6801	1.3011	1.6802	2.0154	2.4141	2.6923
45	0.6800	1.3007	1.6794	2.0141	2.4121	2.6896
46	0.6799	1.3002	1.6787	2.0129	2.4102	2.6870
47	0.6797	1.2998	1.6779	2.0117	2.4083	2.6846
48	0.6796	1.2994	1.6772	2.0106	2.4066	2.6822
49	0.6795	1.2991	1.6766	2.0096	2.4049	2.6800

DOCUMENTACIÓN