Facultad 4

Trabajo de Diploma para optar por el Título de Ingeniero en Ciencias Informáticas

Título: Capa de servicios web para el Juez en Línea Caribeño

Autor(es):

Cesar Andrés González Rodríguez

Omar Almaguer Guerra

Tutor(es):

Ing. Yonny Mondelo Hernández

Ing. Yunier Broche Guevara

Ing. Yelaine Sánchez Oliú

La Habana, 2016.

"Año 58 de la Revolución"

Declaración de autoría

Por este medio declaramos ser los únicos autores del presente trabajo de diploma y reconocemos a la Universidad de las Ciencias Informáticas (UCI) los derechos patrimoniales de la misma, con carácter exclusivo. Autorizamos a dicho centro para que haga el uso que estime pertinente con este trabajo.
Para que así conste se firma la presente a los ____ días del mes de _____ del año _____.

Cesar Andrés González Rodríguez

_____

Firma del Autor

Omar Almaguer Guerra

_____

Firma del Autor

Ing. Yonny Mondelo Hernández

_____

Firma del Tutor

Ing. Yunier Broche Guevara

_____

Firma del Tutor

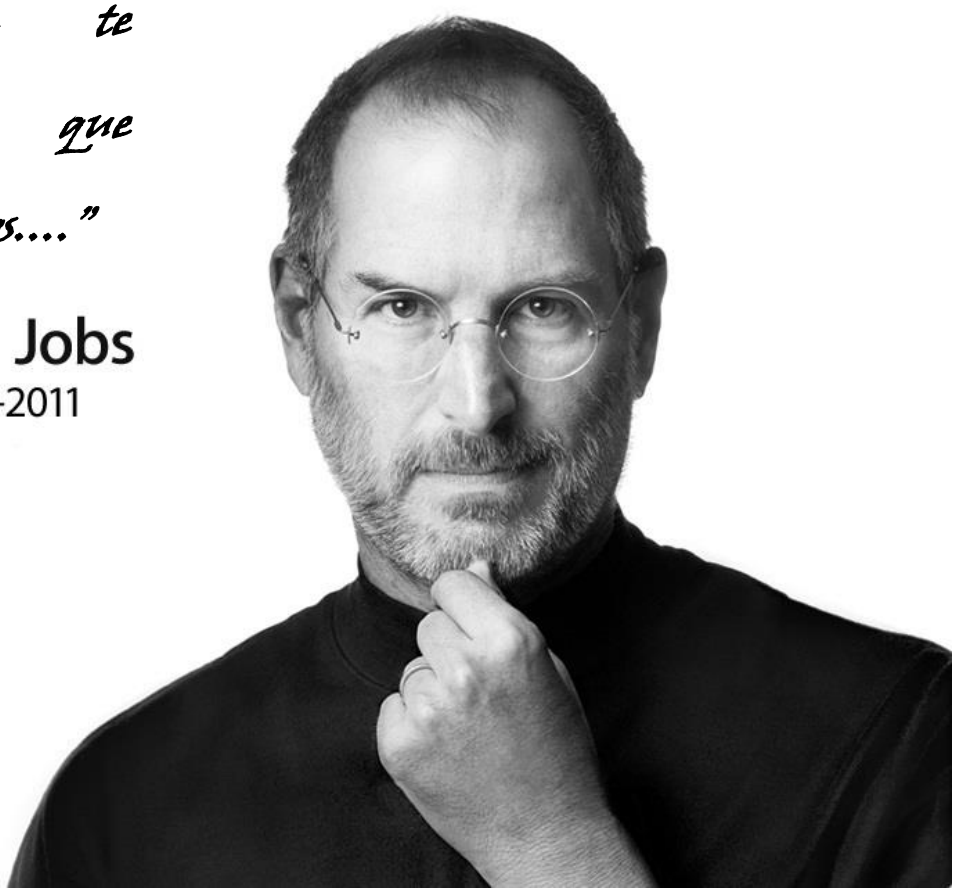
Ing. Yelaine Sánchez Oliú

_____

Firma del Tutor

"Si tú no trabajas por tus sueños, alguien te contratará para que trabajes por los suyos...."

Steve Jobs
1955-2011



Agradecimientos

Agradezco a todas aquellas personas que me apoyaron e hicieron posible la realización del presente trabajo de diploma.

Agradezco especialmente a mi mamá, mi papá, mi abuela, mi padrastro, a mi otra abuela y a toda mi familia en general que siempre confiaron en mí, me guiaron por el camino correcto, me alentaron y ayudaron a seguir adelante con mi sueño.

A mi "chiquitica" por todo su amor y comprensión en estos hermosos años. Por su paciencia y apoyo en tiempos de tesis.

A los tutores: Broche, Yelaine y Yonny, que siempre me ayudaron incondicionalmente en la confección de la tesis.

A los probadores de la aplicación: Osvel y Reysel, y a toda la gente del aula.

De Cesar.

Agradezco el apoyo incondicional de mis padres, a mi hermano Barinoski por su preocupación y servirme de ejemplo en muchas de las decisiones que he tomado. A mi primo Richard Díaz por los años que estuviste aquí en La Habana acompañándome. A todos mis familiares que son pocos y son muchos.

A mis amigos que están dentro y fuera de la UCI, a los que dejaron de ser amigos para convertirse en mis hermanos: Osvaldo, Sifredo, Alejandro, Neobel, Richard. A mis compañeros de aula y compañeros del vicio.

Agradezco a mi compañero de tesis, Cesar, por su increíble paciencia.

A todos los profesores que han influido en mi formación profesional.

A mis tutores por la imprescindible ayuda brindada.

A todos los que me aportaron de una u otra forma algo de conocimiento.

De Omar.

Resumen

Como resultado de la integración de la Universidad de Ciencias Informáticas (UCI) al movimiento ACM-ICPC (ACM-Asociación de los Sistemas Informáticos, ICPC-Competición Internacional Universitaria de Programación), surge el Juez en Línea Caribeño (COJ) con el objetivo de probar, mejorar y compartir habilidades en la resolución de problemas, la programación de computadoras y el trabajo en equipo, además de obtener una fuerte capacitación para participar en competiciones de programación a nivel internacional. El propósito de la presente investigación consiste en la implementación de una capa de servicios web que permita establecer la comunicación de aplicaciones informáticas con el COJ. Esta comunicación permitirá que cualquier sistema autorizado pueda utilizar los servicios web brindados por el juez en línea, contribuyendo a una integración más consolidada con los demás jueces en línea del mundo y otras aplicaciones desarrolladas en cualquier lugar del mundo. Dentro de los servicios disponibles se encuentran: los de obtención de problemas, obtención de envíos, gestión de perfil de usuario, y los de gestión de correos. En la elaboración del presente trabajo, se profundizó en el estudio teórico de la interoperabilidad, los servicios web y el uso de estos en diferentes jueces en línea reconocidos internacionalmente. Una vez concluida la investigación se procedió a desarrollar el diseño de las especificaciones del sistema, luego con una base tecnológica correctamente determinada se inició la implementación de la aplicación. La fase de ejecución y desarrollo finalizó con un ciclo de pruebas para la validación del correcto funcionamiento de la capa de servicios web.

Palabras clave: aplicaciones informáticas, comunicación, juez en línea, servicios web.

Índice

Índice de Ilustraciones

Índice de Tablas

Introducción

Los concursos de programación competitiva fueron iniciados en la década del 70 por el Concurso Internacional Universitario de Programación, fundado y patrocinado por la Asociación de Máquinas Computadoras (ACM-ICPC por sus siglas en inglés), para incentivar a los usuarios en la programación de algoritmos. Este se efectúa anualmente con la participación de varios equipos en representación de diferentes universidades del mundo y constituye en la actualidad uno de los más importantes y prestigiosos eventos en su esfera. Estas competencias han favorecido el desarrollo de habilidades en las técnicas de análisis y diseño de algoritmos, la cooperación en un ambiente de equipo, la productividad y el pensamiento algorítmico, contribuyendo a la formación de mejores profesionales.

En sus inicios este concurso no contaba con muchos participantes, de manera que era posible realizar manualmente la gestión de la información del mismo. En el transcurso de los años, el crecimiento en el número de participantes complejizó el proceso de evaluación, lo cual provocó el surgimiento y evolución de lo que se conoce hoy como jueces en línea.

Un juez en línea es una aplicación web con un conjunto permanente de problemas a resolver. Su principal función es evaluar automáticamente los intentos de solución de los usuarios y ubicarlos en una posición según sus resultados. Su auge se debe al aumento significativo que ha experimentado el movimiento de programación competitiva desde 1997 con la publicación del Juez en Línea de la Universidad de Valladolid (UVA) en España (Europa PRESS, 2012).

En este contexto surge en el 2010 el Juez en Línea Caribeño (COJ, por sus siglas en inglés: Caribbean Online Judge). Esta es una aplicación web desarrollada por la Universidad de las Ciencias Informáticas que actúa como jurado en línea y sirve como herramienta de apoyo al Proceso de Enseñanza y Aprendizaje, particularmente en las asignaturas de programación. Los algoritmos enviados por los usuarios dan solución a disímiles problemas lógicos y matemáticos que se encuentran en el sitio. Los mismos pueden ser enviados en diversos lenguajes como Pascal, Python, C/C++, Java, entre otros. De esta forma se confeccionan las posiciones, ya sea por usuarios, instituciones o países de acuerdo a las puntuaciones obtenidas.

Dichas funcionalidades le han valido reconocimiento internacional contando con más de 24 000 usuarios registrados, más de 3000 problemas y siendo sede de numerosas competencias anualmente como las competencias oficiales del ACM-ICPC en el Caribe (Universidad de la Ciencias Informáticas, 2016).

Debido a esto, varios proyectos han contactado con el COJ para utilizar sus problemas, estadísticas, competencias, hacer envíos de solución a problemas entre otras funcionalidades, pero no ha sido posible ya que estos recursos no están disponibles, al no existir el mecanismo para que las aplicaciones informáticas interactúen con el sitio. Actualmente todas las funcionalidades del sitio incluido el motor de calificaciones son completamente inaccesibles para cualquier sistema automatizado que intente acceder a ellas.

Como consecuencia se limita el acceso desde otras plataformas a las funcionalidades que brinda el COJ; se aísla el proyecto del auge que toman actualmente las tecnologías como el uso de las aplicaciones móviles; se restringe el impacto académico y social del sitio a nivel nacional e internacional y se pierde la oportunidad de hacer que más usuarios conozcan y utilicen el COJ.

A partir de la situación descrita anteriormente se ha identificado el siguiente **problema de investigación**: ¿Cómo lograr la interoperabilidad de las aplicaciones informáticas de terceros con el COJ?

Lo antes expuesto lleva a analizar la comunicación entre sistemas informáticos planteando como **objeto de estudio de la investigación**: la interoperabilidad entre aplicaciones informáticas enfocado en el siguiente **campo de acción**: el proceso de desarrollo de servicios web en jueces en línea.

Para darle solución al problema planteado se tiene como **objetivo general:** Desarrollar una capa de servicios web para lograr la interoperabilidad de las aplicaciones informáticas de terceros con el COJ.

Se define para este trabajo los siguientes **objetivos específicos**:

- Elaborar el marco teórico y el diseño metodológico de la investigación.
- Diseñar la capa de servicios web.
- Implementar la capa de servicios web, de forma que cumpla con los parámetros requeridos.
- Validar la misma mediante pruebas de software.

Para darle cumplimiento a los objetivos antes planteado se proponen las siguientes **preguntas científicas**:

- ¿Cuáles son los presupuestos teóricos y metodológicos que fundamentan el desarrollo de la capa de servicios web para el COJ?
- ¿Cuál es el estado actual del desarrollo de servicios web en jueces en línea?
- ¿Cuáles son las herramientas y tecnologías necesarias para el desarrollo de la capa de servicios web para el COJ?
- ¿Cuáles son las pautas a seguir para definir el diseño de la propuesta a desarrollar?
- ¿Qué tipo de pruebas de software se aplican para validar el correcto funcionamiento de la capa de servicios web para el COJ?

A partir de las preguntas antes planteadas se proponen las siguientes **tareas de investigación**:

1. Analizar los conceptos fundamentales y las principales soluciones existentes referentes a los servicios web que brindan los jueces en línea.
2. Identificar los servicios web que debe ofrecer el COJ.
3. Seleccionar las tecnologías, herramientas y metodología de desarrollo de software que se adapten mejor a las condiciones necesarias para el desarrollo de la capa de servicio web.
4. Definir los requisitos funcionales y no funcionales que debe poseer la capa de servicios web a desarrollar para satisfacer las necesidades existentes.
5. Definir la metodología a utilizar en el desarrollo de la capa de servicios web para el COJ.
6. Elaborar los diagramas, modelos y el diseño de la solución en correspondencia con la metodología seleccionada.
7. Implementar los servicios web que debe ofrecer el COJ junto con la documentación de los mismos.
8. Validar el correcto funcionamiento de la capa de servicios web desarrollada para garantizar el total cumplimiento de los requisitos del proyecto.

Durante el proceso de desarrollo de la capa de servicios web se realizarán investigaciones para profundizar en el objeto de estudio definido, dichas investigaciones serán guiadas por los siguientes métodos teóricos:
Analítico – Sintético: Se aplicó al realizar un estudio acerca de los aspectos relacionados con los servicios web y su implementación, mediante el análisis de la documentación existente con el fin de sintetizar sus principales funcionalidades y ventajas a la hora de ser aplicados.
Histórico – Lógico: Se utilizó para realizar un estado del arte de la problemática analizada con el fin de conocer cómo ha evolucionado la utilización de servicios web hasta la actualidad.

Modelación: Se emplea en la creación de diagramas para representar el proceso de desarrollo mediante el lenguaje de modelado UML para reflejar la estructura, relaciones internas y características de la solución.
Observación: Se utilizó para recopilar información de los jueces en línea existentes, permitiendo la obtención de conocimiento del tema a partir del análisis de las funcionalidades de los mismos.

Con la realización de la presente investigación se pretende obtener el siguiente resultado:

- Una capa de servicios web que permita la interoperabilidad con las aplicaciones informáticas de terceros.
- Documento de tesis de la presente investigación para ser almacenado en el repositorio institucional de la Universidad de las Ciencias Informáticas.

La presente investigación científica cuenta con una estructura de 3 capítulos que pueden ser descritos de la siguiente forma:

Capítulo 1: Fundamentación teórica

Se abordan los conceptos fundamentales que permiten entender los servicios web. Se caracterizan las aplicaciones existentes en la actualidad que responden al problema a resolver y su factibilidad. Así como también, la fundamentación de las herramientas y tecnologías que se emplean en el desarrollo de la presente investigación.

Capítulo 2: Diseño de la solución

Recoge la descripción y análisis de la solución propuesta, la caracterización de los servicios web desarrollados, incluyendo los requerimientos. Se especifican los patrones usados, tanto de diseño como arquitectónicos.

Capítulo 3: Implementación y prueba

Se describen los estándares de código fundamentales usados en el desarrollo de la arquitectura de la capa de servicios web y se generan los artefactos correspondientes a la fase de desarrollo según la metodología seleccionada. Se realizan las pruebas de software que permiten garantizar el cumplimiento de los requisitos de la investigación y se corrobora la hipótesis planteada.

Posteriormente se presentan las conclusiones generales, recomendaciones y referencias bibliográficas utilizadas.

Capítulo 1: Fundamentación Teórica de la capa de servicios web para el COJ.

1.1. Introducción

La sistematización del estudio de los principales conceptos relacionados con la interoperabilidad entre aplicaciones informáticas y los servicios web, permite entender cómo han evolucionado estas tecnologías y conocer cuáles son las últimas tendencias en este campo. Un estudio del desarrollo de los servicios web en el mundo sirve de referencia para conocer cuáles son sus últimas tendencias. Para un buen entendimiento de los servicios web es necesario dominar las herramientas, tecnologías, lenguajes y metodologías de software en que se basan para poder indicar cuáles de estas son las más recomendables para el desarrollo de la propuesta de solución.

1.2. Conceptos asociados al dominio del problema

Un **Juez en Línea** es una aplicación web para entornos generalmente académicos, que incluye varios problemas de distintas materias para ser resueltos mediante técnicas de programación y, lo más importante, evalúa automáticamente las soluciones de sus usuarios en los varios lenguajes de programación disponibles. Se caracterizan por contener una interfaz bien definida para la interacción con el usuario y un alto nivel de disponibilidad, permitiendo el libre acceso a la aplicación en todo momento a las personas registradas (Junco Vázquez, 2012).

A continuación se muestra una ilustración del esquema de funcionamiento general de los jueces en línea.

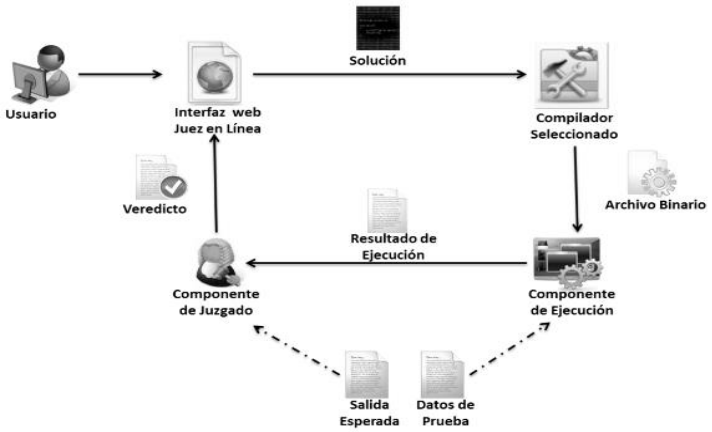


Ilustración 1 Esquema de funcionamiento general de los jueces en línea (Junco Vázquez, 2012).

Según el IEEE (del inglés: Institute of Electrical and Electronics Engineers) se define **interoperabilidad** como la habilidad de dos o más sistemas para intercambiar información y utilizar la información intercambiada (IEEE, 1990). Mientras que otras fuentes definen interoperabilidad como la capacidad que tienen dos o más sistemas informáticos de intercambiar datos entre sí, con el objetivo de utilizarlo para beneficio propio (IDABC, 2009). A partir de estos conceptos se puede llegar a la conclusión de que la interoperabilidad de la información es un aspecto clave en el mundo de las tecnologías de la información ya que a medida que crecen el número de plataformas y mecanismos de comunicación heterogéneos se convierte en un problema lograr el intercambio de datos entre diferentes sistemas informáticos.

La necesidad de que los sistemas informáticos intercambiaran datos entre sí facilitó el surgimiento de los servicios web. Existen múltiples definiciones sobre lo que son los **Servicios Web**, lo que muestra su complejidad a la hora de dar una adecuada definición que englobe todo lo que son e implican. Según el consorcio W3C (del inglés: World Wide Web Consortium) define los servicios web como: “sistemas software diseñados para soportar una interacción interoperable máquina a máquina sobre una red” (W3C, 2014). Mientras por otro lado en investigaciones realizadas por el MSc. Carlos Andrés Morales Machuca de la Universidad de Colombia, plantea que lo servicios web son: “sistemas de software que permiten el intercambio de datos y funcionalidades entre aplicaciones sobre una red, soportando diferentes estándares que garantizan la interoperabilidad de estos sistemas” (MORALES MACHUCA, 2010).

A partir de estas definiciones se puede englobar el término de Servicios Web como un conjunto de aplicaciones o de tecnologías con capacidad para interoperar en la Web. Estas aplicaciones o tecnologías intercambian datos entre sí con el objetivo de ofrecer unos servicios. Los proveedores ofrecen sus servicios como procedimientos remotos y los usuarios solicitan un servicio llamando a estos procedimientos a través de la Web.

Funciones de los Servicios Web (W3C, 2014)**:**

- Proporcionan mecanismos de comunicación estándares entre diferentes aplicaciones.
- Proporcionan interoperabilidad y extensibilidad.
- Posibilidad de combinación entre las aplicaciones para realizar operaciones complejas.

- Se pueden utilizar con HTTP (del inglés: *Hypertext Transfer Protocol*) sobre TCP (del inglés: *Transmission Control Protocol*) en el puerto 80, lo que evita que sean bloqueados por los firewall de la mayoría de los sistemas operativos existentes.

Es relevante señalar que son los estándares abiertos los que emergen como una clave importante para lograr la interoperabilidad (Acero Martín, 2011). Entre los estándares de intercambio de datos destacan por su nivel de preferencia entre desarrolladores: XML (del inglés: Extensible Markup Language) y JSON (del inglés: *Java Script Object Notation*).

JSON: Formato de intercambio de datos ligero, fácil de analizar y generar basado en un subconjunto del lenguaje de programación JavaScript. Es independiente del lenguaje, pero utiliza las convenciones que son familiares para los programadores de la familia de C incluyendo C++ y C#, Java, JavaScript, Perl, Python, etc. (ECMA, 2013).

En la siguiente tabla se muestra un resumen de las principales características de los formatos de intercambios mencionados anteriormente (Boci, y otros, Agosto 2012):

Referencia	JSON	XML
Simplicidad	Tiene una gramática mucho más pequeña que XML y mapea directamente sobre las estructuras de datos utilizadas en el lenguaje de programación.	XML ha sido regularmente criticado por su nivel de detalle y complejidad. El mapeo del modelo de árbol básico de XML hacia los sistemas de tipos de lenguajes de programación o bases de datos puede ser difícil, especialmente cuando se utiliza XML para el intercambio de datos altamente estructurados entre aplicaciones, lo que no era su objetivo primario de diseño.
Extensibilidad	No es extensible, ya que JSON no es un lenguaje de marcado de documentos, por lo que no es necesario definir nuevas	Si es extensible ya que es un lenguaje de marcado de documentos.

	etiquetas o atributos para representar los datos en ella.	
Interoperabilidad	Equivalente entre los dos lenguajes.	Equivalente entre los dos lenguajes.
Código abierto	Equivalente entre los dos lenguajes.	Equivalente entre los dos lenguajes.
Procesamiento	JSON se procesa más fácil ya que su estructura es más simple.	Los analizadores sintácticos de XML no son tan sencillos de crear debido a que la estructura de los datos es compleja.

Tabla 1 Comparación JSON – XML.

Se puede llegar a la conclusión de que XML es difícil de parsear y de leer, su modelo de datos es incompatible con la mayoría de los modelos de datos de los lenguajes de programación y sus ventajas son irrelevantes cuando las necesidades primarias de respuesta son serializaciones de una representación de datos interna.

A continuación se muestra una gráfica de Google Trends que muestra el uso de XML y JSON en las API:

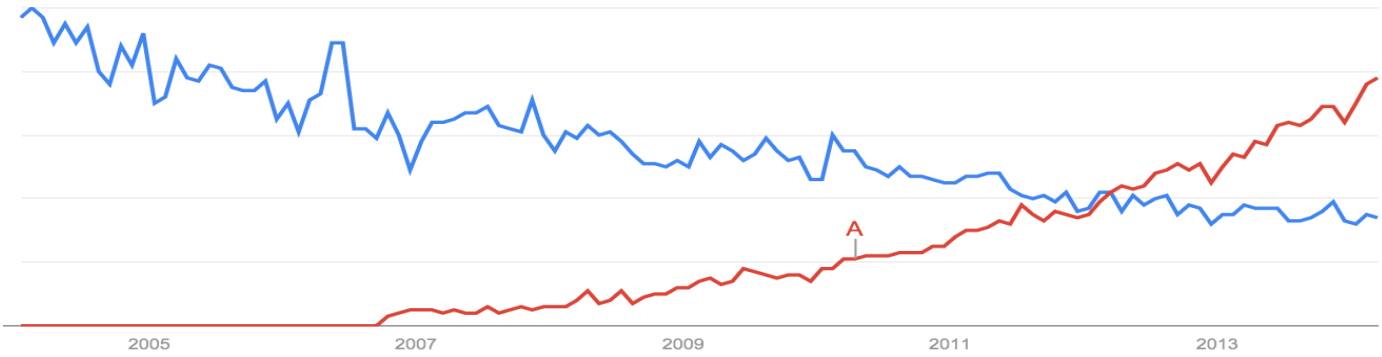


Ilustración 2 Gráfica comparativa de XML vs JSON (Google, 2016).

Por lo anteriormente planteado se escoge como lenguaje de intercambio de datos el formato JSON dada su ligereza, fácil procesamiento y su poca extensibilidad para así optimizar el volumen de intercambio de datos entre las aplicaciones.

Para la correcta implementación de los servicios web es necesaria conocer como está conformada su estructura. Existen un conjunto de servicios y protocolos usados para definir, localizar, implementar y hacer que un servicio web interactúe con otro. Este conjunto es conocido como *Web Services Protocol Stack* y está conformado esencialmente de cuatro subconjuntos (Akhtar, y otros, 2015): servicio de transporte, servicio de mensajería, descripción del servicio, descubrimiento de servicios.

Servicio de transporte: Es el encargado del transporte de los mensajes entre aplicaciones sobre la red (MORALES MACHUCA, 2010). Incluye varios protocolos del nivel de aplicación como son HTTP, FTP (del inglés: *File Transfer Protocol*) y SMTP (del inglés: *Simple Mail Transfer Protocol*).

Servicio de mensajería: Es el conjunto encargado de la codificación de mensajes en un lenguaje estándar para que pueda ser interpretado en cualquiera de los nodos de la red (Akhtar, y otros, 2015). Los componentes más utilizados en este conjunto son los siguientes:

- SOAP (del inglés: *Simple Object Access Protocol*): Es un protocolo de la capa de aplicación para el intercambio de mensajes basados en XML sobre redes de computadoras. Permite utilizar lenguajes de alto nivel para llamar e implementar el servicio web. Es básicamente un paradigma de una sola vía pero con la ayuda de las aplicaciones se puede llegar a crear patrones más complejos. Está constituido por: un marco que describe el contenido del mensaje e instrucciones de proceso; un conjunto de reglas para representar los tipos de datos definidos; convenciones para representar llamadas a procedimientos remotos y respuestas (Weerawarana, y otros, 2005).

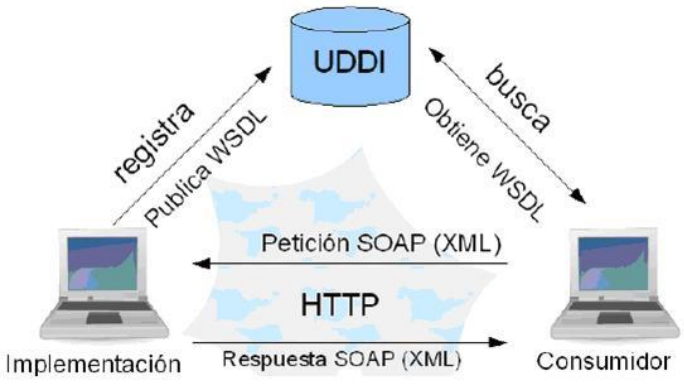


Ilustración 3 Funcionamiento de SOAP (Weerawarana, y otros, 2005).

- REST (del inglés: *Representational State Transfer*): Existen varios conceptos que lo definen como:

"Estilo de arquitectura de software para sistemas hipermedias distribuidos tales como la World Wide Web" (Fielding, 2000).

"Conjunto de principios para el diseño de redes, utilizado comúnmente para definir una interfaz de transmisión sobre HTTP" (MORALES MACHUCA, 2010).

En resumen REST es considerado un conjunto de principios de arquitectura que usa directamente el protocolo HTTP(s) para obtener datos o indicar la ejecución de operaciones sobre los datos en cualquier formato (JSON, XML, etc.). Los principios de REST son (Alvarez, y otros, 2014):

- Stateless o sin estado: Se refiere a que cada mensaje que viaja a través de HTTP(s) lleva toda la información necesaria para realizar una petición, es decir, que el servidor no guarda datos referentes a otras comunicaciones con el cliente (cookies o sesiones).
- Utilizar las operaciones o verbos ya definidos por HTTP: POST, GET, PUT y DELETE.
- Utilización de URI (del inglés: *Uniform Resource Identifier*): Se busca la utilización de URI para identificar de manera precisa un recurso a través de red.
- Hipermedias: El concepto básicamente es que los resultados de los llamados a un sistema basado en REST permitan navegar a otros recursos sin necesidad de hacer alguna transformación extra.
- Escalabilidad: Se puede crecer todo lo que se necesite en cualquier momento. La API REST puede responder a otros tipos de operaciones o puede versionarse tanto como se desee.
- Fiabilidad: Solo hay que tener en cuenta que el nexo cliente/servidor esté correcto. Se pueden hacer cambios en el servidor, lenguajes, bases de datos, etc. y mientras devuelvas los datos que espera el cliente todo funcionará correctamente.

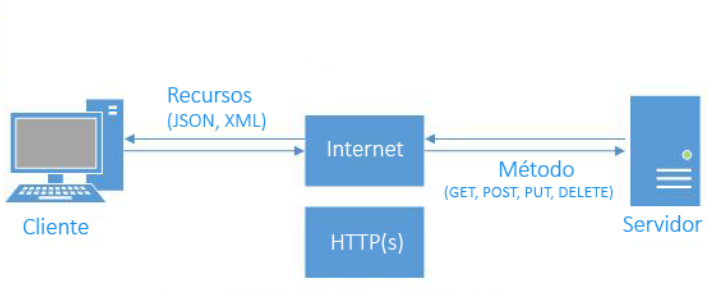


Ilustración 4 Funcionamiento REST.

Dr. Leonard Richardson propuso un modelo de niveles de madurez o complejidad como principios a seguir por los sistemas basados en REST. El modelo describe cuatro niveles:

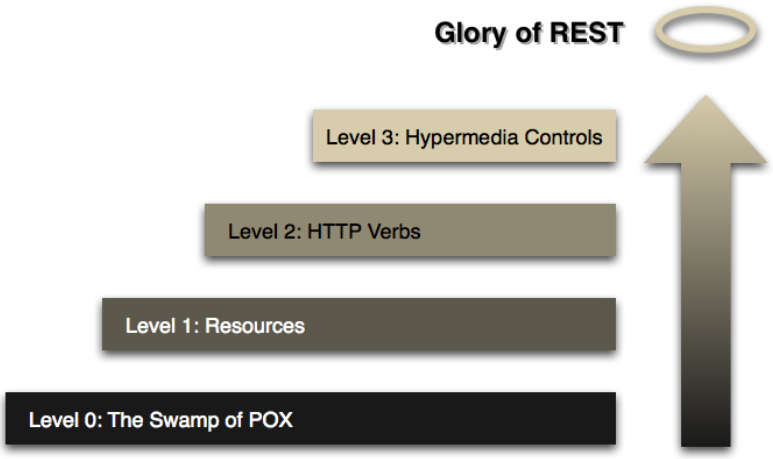


Ilustración 5 Niveles de madurez de sistemas basados en REST (Fowler , 2010).

A continuación se explica en que consiste cada uno de ellos (Fowler , 2010):

- Nivel 0: Se trata de un sistema de estilo RPC sencillo ya que la comunicación está dada por XML simples (POX del inglés: *Plain Old XML*). Si se utilizara SOAP o XML-RPC fuera básicamente el mismo mecanismo, la única diferencia es que se envuelve los mensajes XML en algún tipo de envoltura.
- Nivel 1: La idea es identificar los recursos a través de un URI sin especificar la acción a realizar sobre el mismo. Ejemplo al llamar http:/misitio.cu/personas/1, representa a una persona.
- Nivel 2: Este nivel nos indica que el API debería utilizar los verbos HTTP, mencionados anteriormente. Utilizando correctamente los verbos y las respuestas.
- Nivel 3: Básicamente es utilizar HATEOAS (del inglés: *Hypermedia as the Engine of Application State*), o sea utilizar hipertexto como el mecanismo del estado de la aplicación. Lo que indica HATEOAS es que al realizar una solicitud, el mismo retorne información de cómo trabajar o manipular el recurso mediante más enlaces.

Es recomendable solo implementar los sistemas basados en REST solo hasta el nivel dos ya que a pesar de que la web generalmente trabaja con los principios HATEOAS (ej., cuando se va a una portada y se sigue los hipervínculos que ofrece la página), no están listos estos para las API REST todavía. Cuando se navega

en un sitio, las decisiones sobre qué hipervínculos van a ser clicados se hacen durante la ejecución. Sin embargo, con una API, las decisiones como qué peticiones van a ser enviadas son tomadas cuando el código de integración de la API es escrito, no en tiempo de ejecución. Esto significa que HATEOAS es prometedor pero no está listo para ser protagonista todavía. Aún faltan por definir estándares y herramientas alrededor de este principio para que su potencial sea completamente aprovechado (Sahni, 2015).

La mayor competencia se centra actualmente entre REST y SOAP por ser considerados los estilos arquitectónicos de comunicación más usados. La siguiente imagen tomada del directorio de servicios web muestra el nivel de uso general de los principales protocolos de mensajería para el desarrollo de servicios web a nivel global.

Ilustración 6 Uso general de protocolos de servicio web en aplicaciones (Programmable, 2016).

A continuación se muestra una tabla comparativa entre REST y SOAP (Navarro Marset, 2007) (Francia, 2010):

	REST	SOAP
Características	• Las operaciones se definen en los mensajes. • Una dirección única para cada instancia del proceso. • Cada objeto soporta las operaciones estándares definidas. • Componentes débilmente acoplados.	• Las operaciones son definidas como puertos WSDL. • Dirección única para todas las operaciones. • Múltiple instancias del proceso comparten la misma operación. • Componentes fuertemente acoplados.

Ventajas	• Bajo consumo de recursos. • Las instancias del proceso son creadas explícitamente. • El cliente no necesita información de enrutamiento a partir de la URI inicial. • Los clientes pueden tener una interfaz "listener" (escuchadora) genérica para las notificaciones. • Generalmente fácil de construir y adoptar.	• Fácil (generalmente) de utilizar. • La depuración es posible. • Las operaciones complejas pueden ser escondidas detrás de una fachada. • Envolver API existentes es sencillo. • Incrementa la privacidad.
Desventajas	• Gran número de objetos. • Manejar el espacio de nombres (URI) puede ser engorroso. • La descripción sintáctica/semántica muy informal (orientada al usuario).	• Los clientes necesitan saber las operaciones y su semántica antes del uso. • Los clientes necesitan puertos dedicados para diferentes tipos de notificaciones. • Las instancias del proceso son creadas implícitamente.
Tecnología	• Pocas operaciones con muchos recursos. • Mecanismo consistente de nombrado de recursos (URI). • Se centra en la escalabilidad y rendimiento a gran escala para sistemas distribuidos hipermedia.	• Muchas operaciones con pocos recursos. • Falta de un mecanismo de nombrado. • Se centra en el diseño de aplicaciones distribuidas.
Seguridad	• Se puede implementar sobre el protocolo HTTPS (Hypertext Transfer Protocol Secure) para tener seguridad en las transferencias. • Comunicación punto a punto segura.	• WS-Security. • Comunicación origen a destino segura.

Tabla 2 Comparación entre REST y SOAP.

Después de haber analizado las características, ventajas, desventajas, tecnología y seguridad de los componentes anteriores se decide usar REST, debido a que los sistemas basados en este estilo ofrecen un bajo uso de recursos, alta escalabilidad y rendimiento, así como bajo número de operaciones con muchos recursos.

Descripción del servicio: Es un documento (generalmente en formato XML) que es usado para definir la interfaz pública de un servicio web. Sirve como acuerdo que establece el comportamiento de un servicio web e indica a los clientes cómo interactuar con él (Booth, y otros, 2011).

WSDL (del inglés: Web Services Description Language): Es un tipo de documento XML que describe lo que hace un servicio web, donde se encuentra y la forma de ser invocado. Este provee información muy importante para los desarrolladores, este lenguaje describe el formato de los mensajes que utiliza y a cuales puede responder (Snell, y otros, 2002). Siempre un documento XML WSDL presenta los siguientes elementos (Booth, y otros, 2011):

- Tipos: Tipos de datos usados por los mensajes.
- Mensaje: Que datos son enviados desde un nodo a otro.
- Tipo de puerto: Define las operaciones que pueden ser llamadas.
- Límite: Es la descripción del protocolo que se está utilizando para transportar el mensaje que puede ser HTTP POST, HTTP GET, SOAP y MIME.
- Servicio: Define una colección de puertos (nodos); el puerto especifica una dirección para el límite definiendo así la comunicación para un nodo específico.

WADL (del inglés: Web Application Description Language): Formato basado en XML que proporciona una descripción de aplicaciones web basadas en HTTP. Estas aplicaciones suelen ser servicios web basados en REST (Hadley, 2009). WADL es una tecnología de descripción presentada por la W3C, sin embargo, este consorcio no planea tenerle en cuenta para el desarrollo de sus trabajos. El éxito de este formato se ha visto frenado por WSDL 2.0, ya que este último se puede utilizar para describir servicios web basados en REST.

OpenAPI (anteriormente Swagger): Es una especificación que proporciona una forma de describir, consumir, visualizar y probar los servicios web basados en REST. Una gran variedad de herramientas

pueden generar automáticamente la documentación de los métodos, parámetros, posibles errores y tipos de datos a partir de un fichero que cumpla esta especificación. Los clientes, desarrolladores o no, pueden comprender y consumir los servicios sin tener conocimiento de la forma o lenguaje de implantación de los mismos en el servidor o acceso al código. (Linux Foundation Collaborative Project, 2016).

Para la capa de servicios web se escogió OpenAPI ya que posibilita generar una documentación de los servicios sencilla de comprender para el usuario final y a su vez brinda herramientas con que probar los mismos.

Descubrimiento de servicios: El descubrimiento de servicios Web XML es el proceso consistente en localizar, o descubrir, uno o varios documentos relacionados que describen un servicio Web XML determinado. A través del proceso de descubrimiento, los clientes de servicios Web XML conocen la existencia de un servicio Web XML y dónde encontrar el documento de descripción del mismo (Akhtar, y otros, 2015).

Al igual que con cualquier otro recurso de Internet, no se puede encontrar un servicio Web en particular sin ayuda de algún medio que permita buscarlo. Los directorios de servicios Web proporcionan ubicaciones centralizadas en las que los proveedores de estos servicios pueden publicar información acerca de los servicios de que disponen. Las especificaciones UDDI (del inglés: Universal Description, Discovery and Integration) definen un medio estándar para publicar y descubrir información acerca de los servicios Web. (Microsoft, 2011).

1.3. Servicios web en jueces en línea en el mundo

Es notable el aumento de la cantidad de jueces en línea de relevancia en el mundo que tienen publicadas sus funcionalidades a través de servicios web disponibles en Internet. La tendencia al uso de este tipo de servicios actualmente se realiza mediante la disponibilidad de los recursos a través de las API (del inglés: *Application Programming Interface*). Es oportuno por lo tanto, realizar un estudio sobre algunos ejemplos actuales en los que se integren los jueces en línea con los servicios web.

Juez en línea AIZU.

Creado en el año 2004 por la Universidad de AIZU localizada en Japón, es uno de los jueces en línea que ofrece actualmente una API (*http://judge.u-aizu.ac.jp/onlinejudge/api.jsp*), en este caso basada en REST, donde publica muchas de las funcionalidades internas del juez.

Los servicios web publicados en el sitio están limitados solo a consultar información mediante el método GET del protocolo HTTP por lo que no posee funcionalidades que involucren envío de datos complejos por parte del usuario ni mecanismos de seguridad en los mismos para asegurar la autenticación de los usuarios en el sistema. A continuación se muestra un ejemplo de una consulta al juez en línea para obtener la lista de problemas por volúmenes donde se describe la dirección en que se encuentra situado el recurso, seguido de un conjunto de parámetros que permiten la ejecución de la acción solicitada:

http://judge.u-aizu.ac.jp/onlinejudge/webservice/problem_list?volume=10

Aunque utiliza REST, los servicios que ofrece se encuentran en el nivel cero de madurez según la escala de Richardson ya que estos usan XML como formato de intercambio de datos y no se utilizan sintaxis universales para identificar sus recursos, es decir cada recurso no está direccionado a través de su URI. Además el sitio cuenta con una breve documentación para describir como están constituidas las respuestas y el significado de cada campo de las mismas.

Juez en línea UVa

El jurado en línea de la Universidad de Valladolid es el más antiguo y reconocido sistema de evaluación automática de problemas de programación. Es fuente de problemas en otros jurados, al contar con un amplio banco de estos organizados en volúmenes que se nutren de las competencias de la ACM (Revilla, y otros, 2011). Este jurado no expone directamente sus servicios web sino que lo hace a través de una herramienta que posee llamada uHunt (http://uhunt.felix-halim.net). Además de exponer una API basada en REST, esta herramienta ayuda a los usuarios a encontrar interesantes problemas a resolver y llevar las estadísticas de todos los envíos realizados.

Todas las funcionalidades de dicha herramienta son de consulta de datos por lo que los servicios web expuestos solo se dedican a devolver los recursos solicitados por los clientes externos mediante el método GET. A continuación se muestra un ejemplo de una consulta al servicio web de ver un problema dado su identificador:

http://uhunt.felix-halim.net/api/p/id/36

Se usan las URI para direccionar los recursos solicitados y JSON como lenguaje de intercambio de datos entre el sistema y los clientes externos, por lo que se puede decir que se encuentra en nivel dos de madurez según la escala de Richardson. Para apoyar a los desarrolladores en el uso de la API el sitio cuenta con una documentación donde se explica la forma de consumir los servicios además de poseer un registro de cambios en los mismos para notificar a los desarrolladores de servicios agregados, marcados como obsoletos, modificados o eliminados.

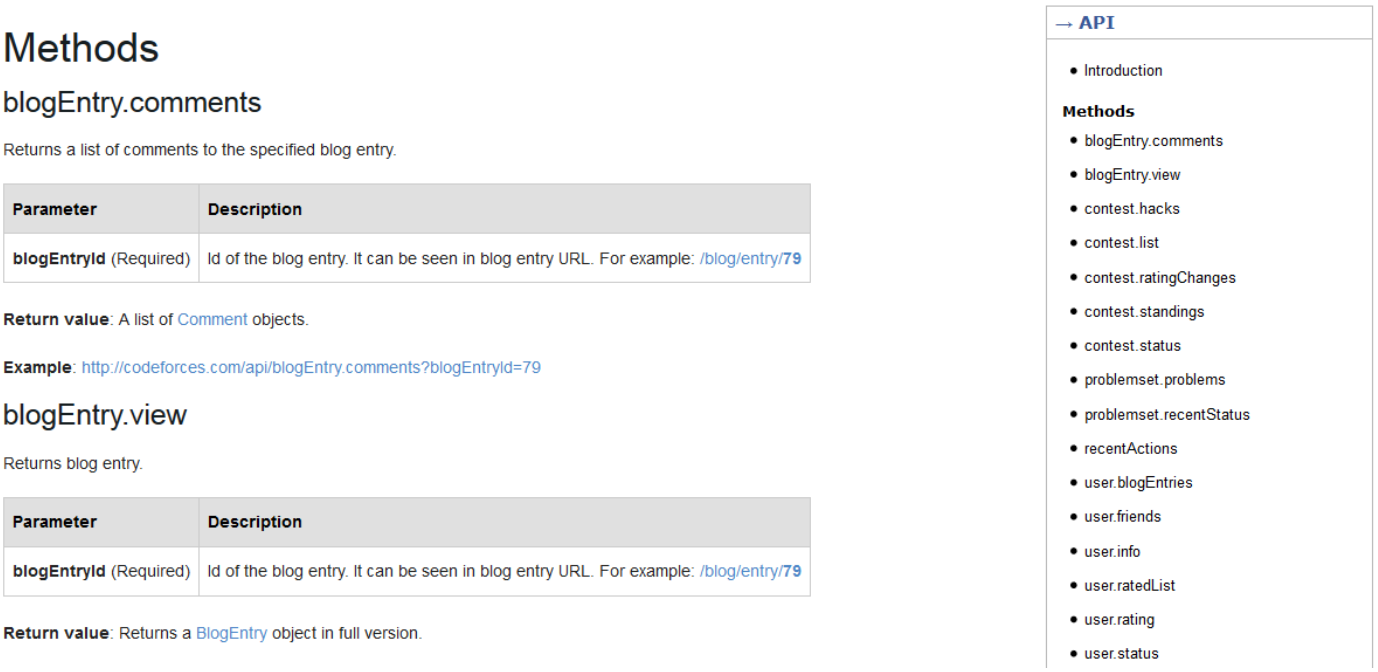
Juez en línea Codeforces

Fue creado en el año 2010 por miembros de la Universidad Estatal de Saratov, Rusia. Actualmente consta de más de 300 000 usuarios registrados, siendo uno de los jueces en línea en el mundo que goza de una alta popularidad. Expone una API basada en REST para exponer las funcionalidades internas del juez a los clientes o aplicaciones externas (http://codeforces.com/api).

Esta API cuenta con varios mecanismos de seguridad para poder manejar y proteger la información sensible de los usuarios registrados y el sistema en general:

- Contiene dos tipos de servicios, los accedidos con autentificación y los que no para así centrar la seguridad en los primeros.
- Para poder utilizar los servicios accedidos con autentificación es necesario el uso de una llave de desarrollador además del envío de un token encriptado con el algoritmo SHA-512 (*del inglés: Secure Hash Algorithm).*
- Para evitar ataques de denegación de servicios solo permite el envío de cinco peticiones por segundos a un servicio web.

Para apoyar a los desarrolladores y clientes externos que quieran utilizar los servicios web el sitio cuenta con una vasta documentación que describe como están constituidas las respuestas y el significado de cada campo de las mismas. De igual forma se describen los parámetros de consulta de las peticiones. A continuación se muestra un ejemplo de la misma:



Methods

blogEntry.comments

Returns a list of comments to the specified blog entry.

Parameter	Description
blogEntryId (Required)	Id of the blog entry. It can be seen in blog entry URL. For example: /blog/entry/**79**

Return value: A list of Comment objects.

Example: http://codeforces.com/api/blogEntry.comments?blogEntryId=79

blogEntry.view

Returns blog entry.

Parameter	Description
blogEntryId (Required)	Id of the blog entry. It can be seen in blog entry URL. For example: /blog/entry/**79**

Return value: Returns a BlogEntry object in full version.

→ API

- Introduction

Methods

- blogEntry.comments
- blogEntry.view
- contest.hacks
- contest.list
- contest.ratingChanges
- contest.standings
- contest.status
- problemset.problems
- problemset.recentStatus
- recentActions
- user.blogEntries
- user.friends
- user.info
- user.ratedList
- user.rating
- user.status

Ilustración 7 Documentación de la API del juez en línea Codeforces (Codeforces, 2016).

Los servicios que ofrece se encuentran en el nivel dos de madurez según la escala de Richardson al usar sintaxis universales para identificar sus recursos y usar los diferentes métodos del protocolo HTTP para acceder a los mismos.

Después de analizar las soluciones existentes en los jueces en línea se reafirmó el uso de servicios web basados en REST ya que se puede apreciar que es la tendencia existente en el mundo para exponer las funcionalidades de este tipo de aplicaciones, además se decidió tener en cuenta a la hora de la construcción de la capa de servicios web para el COJ, las mejores prácticas usadas por estos en la construcción de sus servicios web y seguir las pautas establecidas para alcanzar nivel dos según el modelo de madurez de Richardson en la creación de sistemas basados en REST para mejorar la interacción de las aplicaciones con la capa.

1.4. Metodología de desarrollo

Las metodologías de desarrollo de software surgieron por la necesidad de la industria del software de guiar dicho proceso y eliminar el caos que existía a la hora de creación del mismo. Existen metodologías tradicionales y ágiles, las primeras están pensadas para el uso exhaustivo de documentación durante todo

el ciclo del proyecto mientras que las segundas ponen vital importancia en la capacidad de respuesta a los cambios, la confianza en las habilidades del equipo y al mantener una relación fuerte con el cliente (Expósito, 2008).

La diversidad de centros y proyectos en la UCI hace que la actividad productiva sea cada vez más amplia, y trae consigo la heterogeneidad en el proceso de desarrollo de software. Actualmente el desarrollo de software dentro de la actividad productiva de la UCI se caracteriza por el uso de diferentes metodologías de desarrollo entre robustas y ágiles, específicamente XP, OPEN UP, RUP, BPM, DAC, KIMBALL, SXP, SCRUM, y NOVA OPEN UP. A pesar de la variedad de metodologías usadas, se ha comprobado que muy pocos proyectos la aplican en su totalidad (RODRÍGUEZ SÁNCHEZ, 2015).

Sin importar la metodología que se use en la Universidad, se está planificando con un único cronograma tipo, además de forzar el método de estimación definido, que responde en su gran mayoría a la metodología RUP. Puesto que para erradicar los problemas vinculados con las planificaciones del proyecto y las estimaciones de tiempo, se propuso converger a una única metodología que cubra las particularidades de cada proyecto (RODRÍGUEZ SÁNCHEZ, 2015). Optándose por realizarle una variación al Proceso Unificado Ágil o AUP (por sus siglas en inglés de Agile Unified Process).

De igual manera, existen metodologías específicas basadas en modelos arquitectónicos orientados a servicios, como RUP-SOMA, el Proceso de CBDI-SAE, el Proceso de diseño de Thomas Earl, y otras, que describen, o intentan describir, las fases del ciclo de vida de los servicios en un proyecto con arquitectura orientada a servicios (SOA del inglés: *Service Oriented Architecture*).

1.4.1. RUP-SOMA de IBM

Esta metodología surge de la integración entre el Proceso Racional Unificado o RUP (por sus siglas en inglés de Rational Unified Process) y la metodología SOMA (del Inglés Service Oriented Modeling and Architecture) de IBM. Cubre las áreas de análisis y diseño incluyendo diseño de la arquitectura, no así las de implementación y despliegue de los servicios. Consta de cuatro fases: análisis de transformación empresarial, seguidas por las fases de identificación, especificación y realización que se suceden de manera iterativa. (IBM, 2007)

De forma general el proceso RUP-SOMA de IBM es un proceso bastante preciso para la realización del análisis y diseño de servicios en una SOA. Define actividades, tareas y pasos que unido a su documentación

ayuda a la realización de múltiples tareas. Pero presenta algunas de las siguientes dificultades: es de carácter propietario por lo que no se dispone de su total contenido ni de sus actualizaciones, no se abordan procedimientos para la elaboración de interfaces técnicas del servicio como son WSDL y WADL, entre otros estándares para servicios web e interfaces para servicios con tecnologías Java.

1.4.2. Proceso de CBDI-SAE

Esta metodología cuenta con cuatro disciplinas: Consumo, Provisión, Gestión y Habilitación. Cada una de estas disciplinas se divide en unidades de proceso y estas a su vez en tareas. El proceso de CBDI-SAE cubre el ciclo completo de desarrollo SOA incluyendo actividades de despliegue, monitoreo y gobierno (Allen, 2007). Es una metodología propietaria por lo cual no se tiene acceso al contenido de muchas de sus tareas.

1.4.3. Proceso Unificado Ágil (AUP)

Es una versión simplificada de RUP. Este describe de una manera simple y fácil de entender la forma de desarrollar aplicaciones de software de negocio usando técnicas ágiles y conceptos que aún se mantienen válidos en RUP. El AUP aplica técnicas ágiles incluyendo Desarrollo Dirigido por Pruebas (test driven development - TDD), Modelado Ágil, Gestión de Cambios Ágil, y Refactorización de Base de Datos para mejorar la productividad. Está basada en disciplinas entregables e incrementales con el tiempo, las cuales son: modelado, implementación, prueba, despliegue, administración de la configuración, administración o gerencia de proyecto y entorno.

1.4.4. Agile Unified Process (AUP) UCI

Al no existir una metodología de software universal, ya que toda metodología debe ser adaptada a las características de cada proyecto, exigiéndose así que el proceso sea configurable (RODRÍGUEZ SÁNCHEZ, 2015). Se creó una variación de la metodología AUP, de forma tal que se adapte al ciclo de vida definido para la actividad productiva de la UCI.

De las 4 fases que propone AUP (Inicio, Elaboración, Construcción, Transición) se decide para el ciclo de vida de los proyectos de la UCI mantener la fase de Inicio, pero modificando el objetivo de la misma, se unifican las restantes 3 fases de AUP en una sola, llamada Ejecución y se agrega la fase de Cierre. Propone 7 disciplinas (Modelo, Implementación, Prueba, Despliegue, Gestión de configuración, Gestión de proyecto y Entorno), se decide para el ciclo de vida de los proyectos de la UCI tener 8 disciplinas, pero a un nivel más

atómico que el definido en AUP. Los flujos de trabajos: Modelado de negocio, Requisitos y Análisis y diseño en AUP están unidos en la disciplina Modelo, en la variación para la UCI se consideran a cada uno de ellos disciplinas. Se mantiene la disciplina Implementación, en el caso de Prueba se desagrega en 3 disciplinas: Pruebas Internas, de Liberación y Aceptación y la disciplina Despliegue se considera opcional. Las restantes 3 disciplinas de AUP asociadas a la parte de gestión para la variación UCI se cubren con las áreas de procesos que define CMMI-DEV para el nivel 2, serían CM (Gestión de la configuración), PP (Planeación de proyecto) y PMC (Monitoreo y control del proyecto) (RODRÍGUEZ SÁNCHEZ, 2015).

Para la elección de la metodología se tuvo en cuenta varios factores, algunos heredan directamente de las características que presenta el proyecto COJ ya que es en este donde está enmarcada la capa de servicios web. A continuación se exponen en qué consisten dichos factores:

- Algunas de las metodologías estudiadas no cumplen con los parámetros que requiere la capa de servicios para su implementación y despliegue, como es el caso de RUP-SOMA y el Proceso de CBDI-SAE que presentan una falta de disponibilidad de artefactos y roles debido a su carácter privativo.
- No se requiere de ningún proceso o fase para identificar qué servicios son los candidatos ya que la capa está enfocada en exponer a las aplicaciones de terceros las funcionalidades internas ya creadas y definidas del COJ a través de servicios web.
- Para el desarrollo de software con metodologías como RUP-SOMA o CBDI-SAE es necesario seguir modelos arquitectónicos orientados a servicios.
- Entre los requerimientos del proyecto está obtener como resultado de la presente investigación un conjunto de artefactos y documentación asociada, acorde con la metodología de desarrollo que usa el mismo para evitar heterogeneidad en sus componentes.

Por estas razones se adopta como metodología AUP-UCI, debido a que presenta un enfoque ideal para el desarrollo de la capa de servicios web en el COJ, teniendo en cuenta que esta metodología es la que usa actualmente el proyecto.

1.5. Herramientas y tecnologías a utilizar

Antes de profundizar en el análisis del uso de las herramientas y tecnologías a utilizar se hace necesario entender totalmente el funcionamiento y características principales del COJ que permitan esbozar el panorama sobre el cual está ambientado el desarrollo de la capa de servicios web.

El COJ presenta un desarrollo a la medida, debido a que no se ha tenido en cuenta la necesidad de producir componentes reutilizables separados. En vez de ello se ha priorizado la evolución rápida del sistema y la respuesta a los errores. Ha estado en producción en varias versiones desde el 5 de junio de 2010, fecha en que fue publicado en Internet, estando actualmente en su versión 3.15.13 beta y para su desarrollo son utilizada varias tecnologías y herramientas, de las cuáles resulta necesario para el desarrollo de la capa de servicios web analizar las siguientes:

Entorno de Desarrollo: NetBeans IDE 8.0.1.

Lenguaje de desarrollo: Java 7.

Sistema Gestor de BD: PostgreSQL 9.3.

Herramienta de Modelado: Visual Paradigm for UML 8.0.

Lenguaje de Modelado: Lenguaje Unificado de Modelado 2.0.

Frameworks: Spring Framework 3.5.0.

Servidor Web: Apache-Tomcat 7.0.

1.5.1. NetBeans IDE 8.0.1

NetBeans IDE es un software diseñado para el desarrollo de aplicaciones Java de escritorio, móviles y web. Además, provee un set de herramientas para desarrolladores en PHP y C/C++. Tiene la potencialidad de ser un software libre de código abierto bajo la licencia GPL y tiene una gran comunidad de usuarios y desarrolladores a lo largo del mundo. Incluye un potente editor capaz de analizar el código fuente sintáctica y semánticamente mientras que proporciona plantillas, consejos y generadores de código (NetBeans Community, 2015). Cabe destacar también funcionalidades como el auto-completamiento de código y la integración de frameworks de desarrollo. Proporciona distintas vistas de los datos: desde múltiples ventanas de proyectos hasta herramientas útiles para la creación y gestión eficiente de aplicaciones. Es un software extensible y multiplataforma, ya que puede ser instalado en todos los sistemas operativos que soporten Java como Windows, Linux o Mac. Admite la creación de aplicaciones Web con Java, tiene soporte para el trabajo con el Framework Spring, lo cual resultó beneficioso en el desarrollo del COJ y se acoge también, al desarrollo del presente trabajo.

1.5.2. Java 7

Java es un lenguaje de programación de alto nivel orientado a objetos de propósito general y basado en clases. Su sintaxis se asemeja a la de C y C++, pero elimina sus características menos usadas y más confusas proporcionando de esta forma simplicidad y facilidad de trabajo. Incluye la gestión de almacenamiento automático y es compilado a un conjunto de instrucciones de código de bytes y a formato binario. Cuenta además con potentes entornos de desarrollo como el IDE Netbeans y una abundante documentación tanto en formato duro como digital. Proporciona portabilidad al software implementado ya que el mismo podrá ser ejecutado sobre cualquier plataforma utilizando el JDK (del inglés: Java Development Kit) (Gosling, y otros, 2015). Además, Java constituye una tecnología que permite la construcción de servicios web REST, por lo cual se acoge en el presente trabajo.

1.5.3. PostgreSQL 9.3

PostgreSQL es un sistema de gestión de bases de datos relacional orientado a objetos, libre y gratuito de código abierto publicado bajo la licencia BSD (del inglés: Berkeley Software Distribution). Es compatible con una gran parte del estándar SQL y ofrece muchas características modernas como: consultas complejas, claves foráneas, disparadores, vistas, integridad transaccional, soporte multi-usuario, optimización de consultas y control de concurrencia multi-versión. Es altamente extensible ya que permite la adición de nuevos tipos de datos, funciones, operadores, funciones de agregado, métodos de índice y leguajes procedurales. Se considera un software multiplataforma (Linux, Un ix, BSDs, Mac OS, Beos, Windows) capaz de manejar complejas rutinas y reglas (The PostgreSQL Global Development Group, 2014). Se definió como Sistema Gestor de Bases de Datos a utilizar en el COJ, teniendo en cuenta que es un proyecto de software libre de gran estabilidad.

1.5.4. Visual Paradigm for UML 8.0

Visual Paradigm es una herramienta UML (Unified Modeling Language) profesional que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño Orientados a Objetos, construcción, pruebas y despliegue. El software de modelado UML ayuda a una más rápida construcción de aplicaciones de calidad, mejores y a un menor coste. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación (Visual Paradigm, 2011). Se caracteriza por la disponibilidad en múltiples plataformas (Windows, Linux). Posee características como el diseño centrado en casos de uso y enfocado al negocio que genera un software de mayor calidad. Además de que el modelo y

el código permanecen sincronizados en todo el ciclo de desarrollo. Posee la ventaja que permite la generación de BD (Base de datos), transformación de diagramas de Entidad-Relación en tablas de BD y compatibilidad entre ediciones. Se utiliza en el COJ para la construcción de todos sus diagramas.

1.5.5. Lenguaje Unificado de Modelado 2.0 (UML)

UML es un lenguaje de modelado visual que se usa para especificar, visualizar, construir y documentar artefactos de un sistema de software. Está compuesto por diversos elementos gráficos que se combinan para conformar diagramas. El mismo permite realizar una descripción altamente detallada del sistema, de procesos del negocio, de componentes de software reutilizables, entre otros (Alhir, 2003). Particularmente en el presente trabajo resulta especialmente útil, permitiendo la creación del diagrama de componentes, el diagrama de despliegue del sistema y los diagramas de clases del diseño.

1.5.6. Spring Framework 3.5.0

Spring Framework es una solución ligera para el desarrollo de aplicaciones y contenedor de inversión de control (IoC). Sin embargo, Spring es modular, lo que permite utilizar sólo las piezas que se necesitan. Soporta la gestión de transacciones declarativa, el acceso remoto a su lógica a través de servicios web o RMI, y varias opciones para sus datos persistentes. Ofrece un marco con todas las funciones MVC, y le permite integrar de forma transparente AOP (del inglés: Aspect Oriented Programming) en el software (Johnson, y otros, 2015).

1.5.7. Apache Tomcat 7.0

Tomcat es un Servidor web con soporte de Servlets y JSPs. Incluye el compilador Jasper, que compila JSPs convirtiéndolas en Servlets. Tomcat puede funcionar como servidor web autónomo en entornos con alto nivel de tráfico y alta disponibilidad. Dado que Tomcat fue escrito en Java, funciona en cualquier sistema operativo que disponga de la máquina virtual de Java (The Apache Software Foundation, 2016).

1.5.8. Herramientas para probar los servicios web

Existe un conjunto de herramientas disponibles para probar servicios web. Aunque el propósito de todas ellas es similar, difieren en funcionalidades, características, usabilidad e interoperabilidad. Manteniendo los aspectos anteriormente planteados se seleccionan tres herramientas para compararlas por ser actualmente las más usadas a la hora de probar servicios web.

JMeter 2.8

Es una herramienta de código abierto creada por ASF (del inglés: Apache Software Foundation). Originalmente fue diseñado para probar aplicaciones web, pero se ha extendido a otras funciones de prueba. La función básica de JMeter es cargar pruebas de aplicaciones cliente/servidor, pero también se puede utilizar para la medición del rendimiento. Además, JMeter es también útil en las pruebas de regresión, facilitando la creación de scripts de prueba y comprobando que los cambios realizados no afecten los resultados esperados. Soporta multi-hilos lo que permite lograr concurrencia a la hora de hacer peticiones simultáneas. JMeter ofrece una alta extensibilidad debido al uso de componentes acoplables y ofrece el usuario una amigable interfaz gráfica (GUI). La configuración y creación de planes de pruebas requiere poco esfuerzo y ofrece una serie de informes estadísticos, así como análisis gráfico (Apache JMeter, 2016).

SoapUI 5.2.1

SoapUI es una herramienta de código abierto para probar sistemas SOA y servicios web en general. Es desarrollado por SmartBear y se distribuye libremente bajo la licencia GNU/LGPL. Facilita la rápida creación de pruebas de rendimiento y ejecución de pruebas funcionales automatizadas. El conjunto de características que ofrece SoapUI ayuda en la evaluación del desempeño de los servicios web. El análisis de los resultados de las pruebas proporciona un medio para mejorar la calidad de los servicios y aplicaciones. Ofrece una interfaz gráfica de fácil uso (SmartBear Software, 2016).

Storm

Storm es una herramienta gratuita y de código abierto para realizar pruebas a los servicios web. Storm es desarrollado en lenguaje F# y está disponible para uso gratuito, distribuido bajo la licencia New BSD. Storm permite probar los servicios web implementados en cualquier tecnología (.Net, Java, etc.). Storm soporta invocaciones dinámicas de métodos de servicios web, incluso los que tienen parámetros de entrada de tipos de datos complejos y también facilita la edición/manipulación de las solicitudes SOAP. La interfaz gráfica de usuario es muy simple y fácil de usar. Múltiples servicios web pueden ser probados a la vez para ahorra tiempo (Araojo, 2016).

A continuación una tabla comparativa de estas herramientas para probar servicios web (Hussain, y otros, 2013):

Características	JMeter	SoapUI	Storm
Tecnologías que soporta	Web-HTTP, HTTPS, SOAP, Database via JDBC, JMS, REST, LDAP, Mail-SMTP, POP3, IMAP	Web-HTTP, HTTPS, SOAP, Database via JDBC, JMS, REST, AMF	SOAP
Primera liberación	2001	2005	2008
Última Versión	2.13	5.2.1	1.1
Lenguaje de desarrollo	Java	Java	F#
Sistema operativos que soporta	Multiplataforma	Multiplataforma	Windows 7, 8, 8.1, 10
Requerimientos	JRE1.5+	JRE1.6+	.NET Framework 2.0 F# 1.9.3.14

Tabla 3 Herramientas para probar servicios web.

Se escoge JMeter debido a que esta herramienta se adapta perfectamente a las características de la capa de servicios web y soporta las pruebas de carga y stress de los servidores web, además de contar con rápidos tiempos de respuestas y madurez en cuanto a su desarrollo. Se puede utilizar en el sistema operativo Linux, entorno donde se desarrolla la capa de servicios.

1.6. Conclusiones parciales

- A través del análisis de los principales conceptos se logró elaborar una panorámica en cuanto al objeto de estudio de la investigación donde se observa principalmente la estructura de los servicios web y los elementos que lo componen.
- Los servicios web REST son la forma escogida para lograr la interoperabilidad entre el COJ y las aplicaciones informáticas de terceros.
- Se eligió AUP-UCI como metodología para garantizar la completitud y calidad del software a la hora de guiar eficientemente no solo el proceso de desarrollo de software, sino también la organización, gestión, configuración, cambios y el soporte de la capa de servicios web.
- El análisis de las principales características de las herramientas y tecnologías adoptadas, enfocadas en las ventajas y facilidades de cada una de estas, permitió crear un marco de trabajo adecuado a las necesidades y características propias de la capa de servicios web para el COJ.

Capítulo 2: Diseño de la propuesta de solución

2.1. Introducción

Teniendo en cuenta las necesidades reales presentadas en la situación problemática de este trabajo de diploma, se decide desarrollar una capa de servicios web para el Juez en Línea Caribeño que exponga las funcionalidades ya implementadas en el COJ, dentro de la capa de servicios, para que puedan ser accedidas por aplicaciones de terceros. Para el desarrollo de los diferentes servicios de la capa es de suma importancia tener en cuenta el funcionamiento de los servicios REST, así como la interoperabilidad de estos.

El objetivo principal de este capítulo es modelar una solución teniendo en cuenta los requerimientos del sistema, así como emplear una arquitectura eficaz para la organización de sus componentes. Para esto será necesario generar una serie de artefactos ingenieriles que esquematicen como estará estructurado el diseño de la capa de servicios para su posterior implementación.

2.2. Propuesta del sistema

La capa de servicios estará enfocada en permitirle a las aplicaciones que interactúen con ella, emplear las funcionalidades ya implementadas del COJ a través de peticiones REST mediante el protocolo HTTP(s), ofreciendo como resultado un fichero JSON con la respuesta de la petición. La siguiente ilustración muestra una idea gráfica de lo que se propone como sistema general:

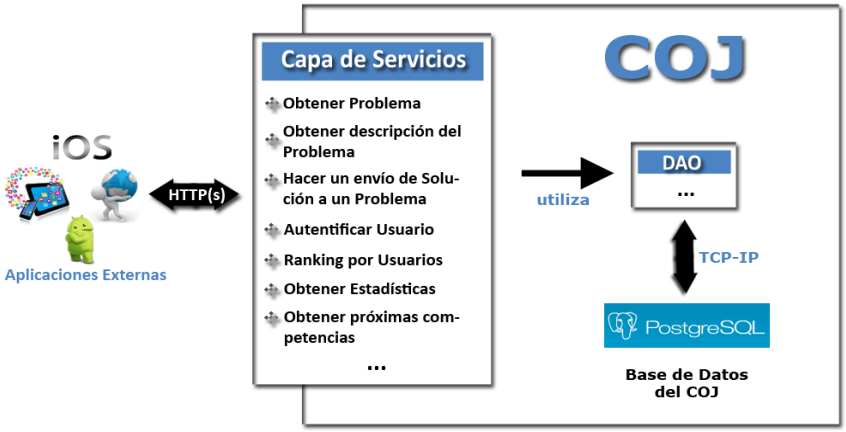


Ilustración 8 Propuesta del Sistema.

Para que otras aplicaciones en diferentes entornos de desarrollo puedan utilizar estas funcionalidades se desarrollará una API (del inglés: Application Programming Interface). Esta proporcionará una serie de funciones que darán mayor abstracción a los desarrolladores y servirá como documentación a los mismos facilitando el uso de la capa de servicios web.

Después de estudiadas las estadísticas de acceso de los usuarios a las funcionalidades del COJ mediante los “logs” del sistema se decidió llevar a servicio web las siguientes por su uso:

Grupo	Servicios Web
Filtros	• Obtener todas las clasificaciones de los problemas. • Obtener todos los jueces en línea. • Obtener los lenguajes habilitados en el COJ. • Obtener las traducciones disponibles. • Obtener los veredictos. • Obtener los lenguajes habilitados de un problema en específico
Sistema	• Recuperar la contraseña olvidada de un usuario. • Generar una llave de desarrollador (No se crea a partir de una funcionalidad). • Autenticar a un usuario en el sistema.
Problema	• Obtener todos los problemas del archivo 24 horas • Obtener todos los problemas del archivo competencias. • Obtener todos los problemas por páginas. • Actualizar el estado de favorito de un problema. • Ver la descripción de un problema.
Extras	• Obtener las competencias que se efectuarán próximamente en diferentes jueces en línea • Postear un comentario en el COJ. • Obtener las preguntas y respuestas frecuentes del COJ (FAQs).
Envíos	• Obtener todos los envíos de solución que han realizado los usuarios en el archivo 24 horas.

	• Obtener todos los envíos de solución que han realizado los usuarios en el archivo de competencia dado el identificador de la misma. • Obtener los mejores envíos de solución realizados a un problema en específico. • Permitir a un usuario hacer un envió de solución a un problema.
Competencias	• Obtener todas las competencias actualmente en ejecución. • Obtener todas las competencias pasadas. • Obtener todas las competencias que próximamente se efectuarán.
Posiciones	• Obtener la tabla de posiciones de los usuarios del COJ. • Obtener la tabla de posiciones de los países registrados en el sistema. • Obtener la tabla de posiciones de las instituciones registradas en el sistema. • Obtener la tabla de posiciones de los usuarios de acuerdo a sus puntuaciones en las competencias. • Obtener la tabla de posiciones de las instituciones, filtradas por el país donde pertenecen. • Obtener la tabla de posiciones de los usuarios, filtrados por el país donde pertenecen. • Obtener la tabla de posiciones de los usuarios, filtrados por la institución donde pertenecen.
Correo	• Eliminar un correo interno de un usuario registrado en el COJ. • Enviar correo interno a un usuario. • Obtener todos los correos almacenados en la bandeja de entrada de un usuario. • Obtener todos los correos almacenados en la bandeja de salida de un usuario. • Obtener todos los correos almacenados en la bandeja de borradores de un usuario. • Cambiar el estado de leído/no-leído a un correo interno.
Perfil de Usuario	• Modificar perfil de usuario

	• Ver el perfil de un usuario.
Estadísticas	• Obtener todas las estadísticas del archivo 24 horas. • Obtener todas las estadísticas del archivo competencias. • Comparar dos usuarios registrados del COJ. • Obtener todas las estadísticas de un problema en específico.

Tabla 4 Funcionalidades a exponer mediante la capa de servicios web.

2.3. Modelo de Dominio

Un modelo del dominio es una representación de las clases conceptuales del mundo real, no de componentes software. Muestra clases conceptuales significativas en un dominio del problema; es el artefacto más importante que se crea durante el análisis orientado a objetos (Craig, 2003).

A continuación se muestra el modelo de dominio que representa la capa de servicios web para el COJ:

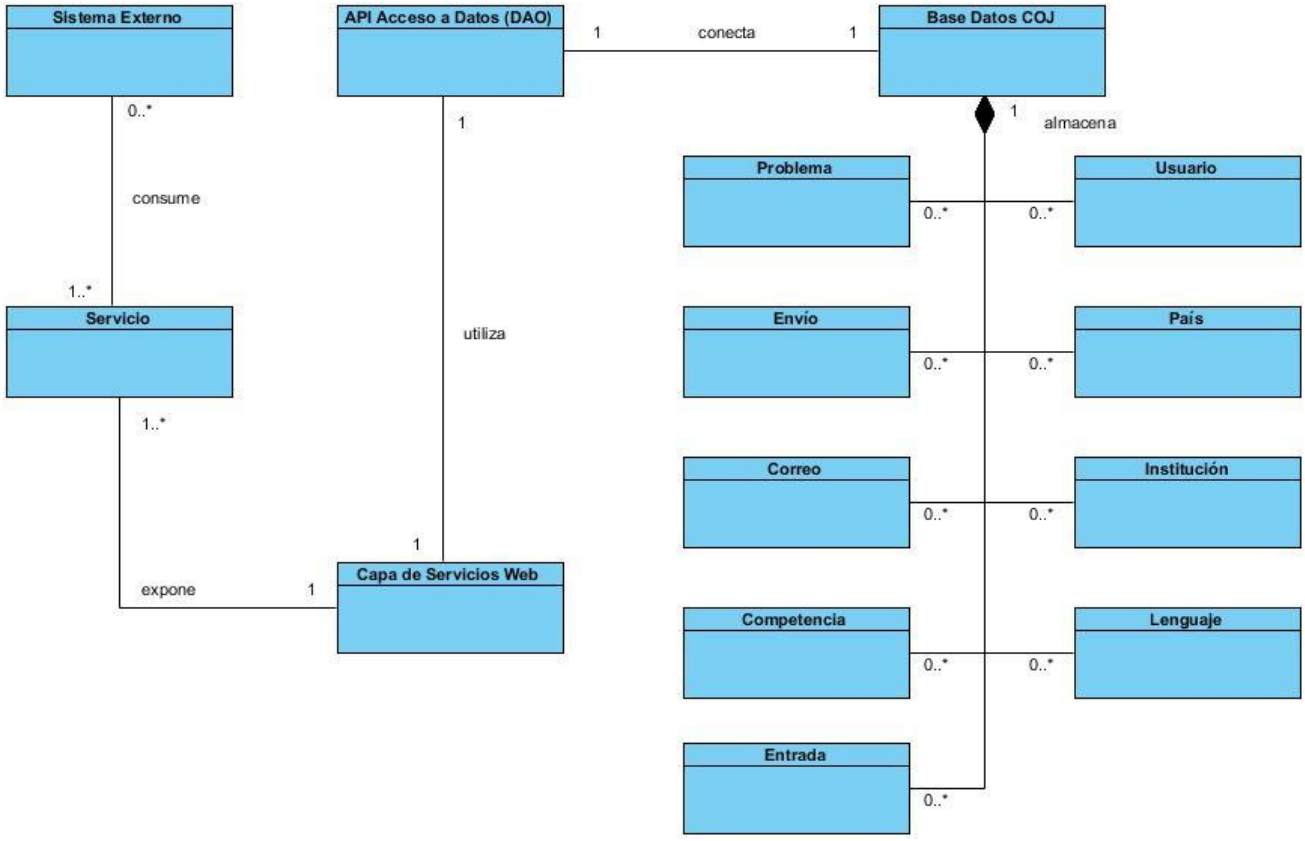


Ilustración 9 Modelo de Dominio.

2.3.1. Descripción de las clases del Modelo de Dominio

A continuación se explica en que consiste cada una de las clases del Modelo de Dominio:

- **Servicio:** Funcionalidad que es ofrecida para ser utilizada por los sistemas externos.
- **Sistema Externo:** Aplicación informática que va a consumir los servicios web que se brindan.
- **Base de Datos:** Aplicación informática que almacena toda la información generada del COJ de forma persistente.
- **Capa de Servicio Web:** Sistema que expone los servicios web del COJ, atiende las solicitudes que recibe generando respuestas a estas. Interactúa con la API de acceso a Datos.
- **API de Acceso a Datos:** Capa de abstracción creada en el COJ que permite la traducción del modelo relacional al modelo orientado a objeto sin necesidad de ORM lo que facilita el acceso y persistencia de los datos.
- **Problema:** Hace referencia a todos los problemas que existen en el COJ a los que los usuarios le hacen envíos de soluciones.
- **Envío:** Información relacionada con los envíos de soluciones que son juzgados posteriormente por el motor de calificaciones.
- **Usuario:** Clientes autorizados a consumir los servicios web privados del COJ, ya que los públicos pueden ser accedidos sin autentificación.
- **Institución:** Hace referencia a las disímiles instituciones que tienen varios competidores que la representan.
- **País:** Información de los países que tienen competidores que los representan en los eventos del COJ.
- **Lenguaje:** Diferentes lenguajes de programación en los que se hacen los envíos de solución a los problemas.
- **Correo:** Mensaje de comunicación interna dentro del COJ.
- **Competencia:** Información relacionada con las competencia que ocurren ya sean pasadas, en ejecución o futuras.
- **Entrada:** Últimas entradas de los usuarios, comentarios u opiniones de diferentes personas dentro del COJ.

2.4. Relación de los requerimientos del sistema

Los requisitos funcionales son declaraciones de los servicios que debe proporcionar el sistema de manera que este reaccione a entidades particulares y de cómo se debe comportar en situaciones específicas (UCID: SOFTWARE-DEFENSA, 2012).

2.4.1. Requerimientos funcionales:

RF1. Listar filtro de clasificaciones de los problemas.

RF2. Listar jueces en línea.

RF3. Listar lenguajes habilitados.

RF4. Listar traducciones habilitadas.

RF5. Listar veredictos.

RF6. Listar lenguajes habilitados dado un problema.

RF7. Recuperar contraseña olvidada.

RF8. Generar una llave de desarrollador.

RF9. Autentificar un usuario.

RF10. Listar problemas del archivo 24 horas.

RF11. Listar problemas del archivo de competencias.

RF12. Listar problemas del archivo 24 horas por páginas.

RF13. Agregar/Quitar problema como favorito.

RF14. Ver descripción del problema.

RF15. Listar próximas competencias de diferentes jueces en línea.

RF16. Listar próximas competencias dado el juez en línea.

RF17. Postear Entrada.

RF18. Obtener últimas entradas de los usuarios por página.

RF19. Obtener preguntas frecuentes (FAQs).

RF20. Listar envíos del archivo 24 horas.

RF21. Listar los mejores envíos de un problema.

RF22. Listar envíos del archivo de competencias.

RF23. Listar envíos del archivo de competencias por páginas.

RF24. Obtener la última página de envíos.

RF25. Listar envíos del archivo 24 horas por páginas.

RF26. Listar una cantidad específica de envíos del archivo 24 horas.

RF27. Hacer un envío de solución a un problema.

RF28. Listar próximas competencias.

RF29. Listar competencias previas.

RF30. Listar competencias actualmente en funcionamiento.

RF31. Ver descripción de competencia.

RF32. Listar posiciones de los usuarios en las competencias.

RF33. Listar posiciones de los usuarios en las competencias por página.

RF34. Listar posiciones de los países.

RF35. Listar posiciones de los países por páginas.

RF36. Listar posiciones de las instituciones.

RF37. Listar posiciones de las instituciones por páginas.

RF38. Listar posiciones de los usuarios.

RF39. Listar posiciones de los usuarios por páginas.

RF40. Ver descripción de país.

RF41. Ver descripción de institución.

RF42. Listar posiciones de las instituciones por país.

RF43. Listar posiciones de los usuarios por país.

RF44. Listar posiciones de los usuarios por institución.

RF45. Eliminar un correo.

RF46. Listar bandeja de borradores.

RF47. Listar bandeja de entrada.

RF48. Listar bandeja de salida.

RF49. Enviar un correo.

RF50. Modificar Perfil de Usuario.

RF51. Ver perfil de usuario.

RF52. Listar estadísticas del archivo 24 horas.

RF53. Comparar Usuarios.

RF54. Listar estadísticas del archivo competencias.

RF55. Listar estadísticas de un problema.

2.4.2. Requerimientos no funcionales:

Son propiedades o cualidades que el producto debe tener que lo hacen atractivo, usable, rápido o confiable. Estos requisitos pueden marcar la diferencia entre un producto aceptado y otro con poca aceptación, no definen el éxito general del producto, pero sí influyen en la evaluación del cliente (Sommervilles, 2005).

Se definieron una serie de categorías para englobar las cualidades que debe tener la capa de servicio web para su correcto funcionamiento y prestación adecuada de sus funcionalidades. A continuación se describen los requerimientos no funcionales:

Software del Servidor:

RNF1. GNU/Linux como sistema operativo.

RNF2. Máquina Virtual de Java.

Hardware del Servidor:

RNF3. Requiere de la utilización de dos servidores como mínimo. Uno para la aplicación web que maneja toda la carga de los usuarios conectadas a este y otro para el motor de calificaciones que se encarga de evaluar automáticamente los envíos de solución a los problemas.

RNF4. Requieren ambos servidores, como mínimo, 1 GB (Giga Bytes) de RAM (Random Access Memory).

RNF5. Como mínimo de CPU es necesario Dual-Core a 2.00 GHz.

RNF6. El disco duro requiere, como mínimo, 5 GB para almacenar la Base de Datos.

Seguridad:

RNF7. Los métodos que requieren de autentificación en la capa de servicios deben estar disponibles solo para aquellos usuarios correctamente autenticados.

RNF8. Por las características del sistema, se ha definido que se devuelve un token al autenticar un usuario y que el tiempo límite de demora que tiene el consumidor para solicitar un servicio antes de que expire su token es de 15 minutos, pasado ese tiempo tendría que volver a obtener uno nuevo.

RNF9. Cada desarrollador que utilice la capa de servicios web tendrá que generar una llave de desarrollador (“apikey”) que lo identificará en el sistema.

Portabilidad:

RNF10. Independientemente de la plataforma de desarrollo en la que fue implementada la capa de servicios web, las aplicaciones que la consuman pueden hacerlo sin tener en cuenta el lenguaje de programación o la plataforma en que fue desarrollada.

Rendimiento:

RNF11. La capa de servicios debe ser capaz de responder una petición en el menor tiempo posible dependiendo del tipo de petición y los datos que se manejan en esta.

Soporte:

RNF12. Se requiere una documentación apropiada que describa todas las funcionalidades del sistema desarrollado, así como una guía para su uso.

Flexibilidad:

RNF13. El sistema permitirá que se defina mediante parámetros las diferentes formas en las que se realizarán las peticiones.

Disponibilidad:

RNF14. La aplicación debe estar disponible en todo momento. Se les garantizará a los usuarios el acceso a la información.

2.5. Modelación a través de Historia de Usuarios

A partir de las características de la capa de servicio web se propone la modelación de esta a través de Historias de Usuario (HU) según lo planteado en la metodología AUP UCI. Tienen el mismo propósito de los casos de uso ya que expresan su punto de vista en cuanto a las necesidades del sistema. Son descripciones cortas y escritas en el lenguaje del usuario sin terminología técnica y proporcionan los detalles sobre la estimación del riesgo y cuánto tiempo conllevará su implementación. Su nivel de detalle debe ser el mínimo posible, de manera que permita hacerse una ligera idea de cuánto costará implementar el sistema. (Yunior Mesa Reyes, 2012)

A continuación se presentan algunas de las HU:

Historia de Usuario	
Número: 9	**Nombre del requisito:** Autenticar usuario.
Programador: Omar Almaguer Guerra	**Iteración Asignada:** 1

Prioridad: Alta.	**Tiempo Estimado:** 1 semana.
Riesgo en Desarrollo: Riesgos definidos en el proyecto COJ.	**Tiempo Real:** 5 días.
Descripción: **1- Objetivo:** Permitir autenticar al usuario en el sistema y devolver un token para poder acceder a los servicios con autentificación a la capa de servicios web. **2- Acciones para lograr el objetivo (precondiciones y datos):** - El usuario debe haberse registrado anteriormente en el sistema, es decir ser usuario del sistema. **3- Comportamientos válidos y no válidos (flujo central y alternos):** El sistema devuelve los siguientes datos si el usuario y la contraseña son correctas: - Token de usuario - Tiempo en que expira. Se devuelve el error HTTP 401 UNAUTHORIZED si el usuario o la contraseña son incorrectos. **4- Flujo de la acción a realizar:** Consultar la URI *POST coj.uci.cu/api/private/login*	

Tabla 5 HU Autenticar Usuario.

Historia de Usuario	
Número: 10	**Nombre del requisito:** Listar problemas del archivo 24 horas.
Programador: Cesar A. González	**Iteración Asignada:** 1
Prioridad: Alta.	**Tiempo Estimado:** 3 días.
Riesgo en Desarrollo: Riesgos definidos en el proyecto COJ	**Tiempo Real:** 2 días.
Descripción:	

1- Objetivo:

Se encarga de devolver al sistema que realizó la petición, un mensaje que incluye todos los problemas disponibles del COJ.

2- Acciones para lograr el objetivo (precondiciones y datos):

3- Comportamientos válidos y no válidos (flujo central y alternos):

El sistema devuelve los siguientes datos:

- Identificador
- Título
- Envíos
- Aceptados
- Puntuación

El sistema devuelve los siguientes datos adicionales si se suministró correctamente el token y la llave de desarrollador:

- Estado de favorito
- Estado de resolución del problema

4- Flujo de la acción a realizar:

Consultar la URI *GET coj.uci.cu/api/problem*

Tabla 6 HU Listar problemas del archivo 24 horas.

Historia de Usuario	
Número: 27	**Nombre del requisito:** Hacer un envío de solución
Programador: Cesar A. González	**Iteración Asignada:** 1
Prioridad: Alta.	**Tiempo Estimado:** 2 semanas.
Riesgo en Desarrollo: Riesgos definidos en el proyecto COJ.	**Tiempo Real:** 2 semanas.
Descripción: **1- Objetivo:** Hace un envío a un problema para que el juez lo califique.	

2- Acciones para lograr el objetivo (precondiciones y datos): - El usuario debe estar autentificado. **3- Comportamientos válidos y no válidos (flujo central y alternos):** Se crea un nuevo envío en el sistema y se mueve hacia el servidor RabbitMQ. El sistema devuelve los siguientes datos si se suministró correctamente el token de usuario, la llave de desarrollador, código fuente y el identificador del envío: - Identificador del envío - Identificador del problema - Identificador del lenguaje en que se realizó el envío - Usuario Se devuelve el error HTTP 412 PRECONDITION FAILED si código fuente es incorrecto. Se devuelve el error HTTP 404 NOT FOUND si el identificador del problema es incorrecto. Se devuelve el error HTTP 401 UNAUTHORIZED si el token de usuario es incorrecto. Se devuelve el error HTTP 429 TOO MANY REQUEST si el usuario supera el límite de solicitudes permitidas. **4- Flujo de la acción a realizar:** Consultar la URI *POST coj.uci.cu/api/judgment/submit*

Tabla 7 HU Hacer un envío de solución.

2.6. Modelo de Diseño

El modelo de diseño juega un papel importante en el desarrollo de software. Permite producir varios modelos del sistema o producto que se va a construir, dicho modelo forma una especie de plan para la solución que será generada. Estos modelos son evaluados en relación con su calidad y pueden ser mejorados antes de generar código y realizar pruebas (Macas, 2009).

2.6.1. Arquitectura de la capa de servicios

La arquitectura del software determina la estructura general del mismo y las formas en que proporciona una integridad conceptual para un sistema. En su forma más simple, la arquitectura es la organización de los componentes del programa, la manera en que estos interactúan y la estructura de datos que utilizan. En un

sentido más amplio, los componentes pueden generalizarse para representar elementos importantes del sistema y sus interacciones (Sommervilles, 2005).

El proyecto COJ usa Modelo Vista Controlador (MVC del inglés*: Model View Controller*) como arquitectura del sistema, más la utilización de una capa auxiliar de acceso a los datos construida sobre Spring que permite la traducción del modelo relacional al modelo orientado a objeto sin necesidad de ORM (del inglés: *Object Relational Mapping*).

La capa de servicios se implementará bajo el diseño arquitectónico del COJ, usando sus funcionalidades ya implementadas y sus diferentes capas. Para el desarrollo de la misma se decidió emplear la arquitectura de software ***n-capas*** específicamente cuatro capas: presentación, controlador, acceso a datos (DAO del inglés: Data Access Object) y modelo. A continuación se muestra una figura con la arquitectura de la capa de servicios web:

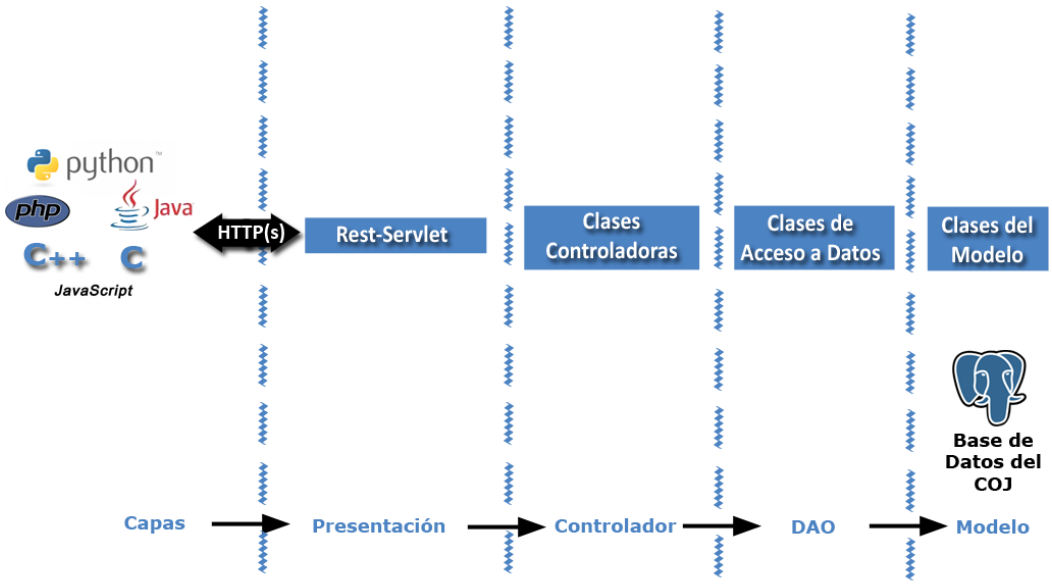


Ilustración 10 Arquitectura de la capa de servicios del COJ.

Esta arquitectura se basa en una distribución jerárquica de los roles y las responsabilidades para proporcionar una división efectiva del problema a resolver. Separa de forma clara la funcionalidad de cada capa. Tiene como principal ventaja que el desarrollo se puede llevar a cabo en varios niveles y, en caso de

que sea necesario realizar algún cambio, solo se corrige el nivel requerido sin tener que revisar todo el código (Sommervilles, 2005).

2.6.2. Patrones de Diseño

Un patrón de diseño describe una estructura de diseño que resuelve un problema de diseño dentro de un contexto específico y en medio de fuerzas que pueden tener un impacto en la manera en que se aplica y utiliza el patrón (Larman, 2001).

Patrones para asignar responsabilidades GRASP:

Controlador: Se utiliza en la capa de servicios a la hora de controlar los datos recibidos por el usuario para después decidir qué métodos son llamados en las distintas clases controladoras. Se utilizan diferentes clases controladoras conteniendo diferentes servicios web con similitudes en cuanto a los recursos que son mostrados. A continuación se muestra un fragmento del código donde se pone de manifiesto como cada método dentro de una clase controladora "sabe" cuando es llamado a través de su URI:

```
@Controller
@RequestMapping("/extras")
public class RestExtrasController {
      @RequestMapping(value = "/faq", method = RequestMethod.GET)
      public List<Faq> ListarFaqs() {...}
}
```

Bajo Acoplamiento: Debe haber pocas dependencias entre las clases. Se pone de manifiesto a la hora de la construcción de la mayoría de las clases, solo dependen estas de la capa inferior y al dividir los eventos del sistema en el mayor número de clases controladoras. A continuación se muestra un fragmento del código donde se pone de manifiesto como las clases controladoras solo dependen de su capa inferior, en este caso acceso a datos (DAO):

```
@Controller
@RequestMapping("/extras")
public class RestExtrasController {

 @Resource
 private EntryDAO entryDAO;
 @Resource
 private UserDAO userDAO;
 @Resource
```

```
    private WbSiteDAO wbSiteDAO;
    @Resource
    private UtilDAO utilDao;

    ...
}
```

Controlador Frontal: Al estar la capa de servicios web construida en Spring Framework se usa un Rest-Servlet para asegurar que las solicitudes entrantes sean enviadas a los controladores correspondientes. A continuación se muestra un fragmento del código donde se pone de manifiesto la redirección de las peticiones hacia la capa de servicios web en el controlador frontal:

```
<servlet>
<servlet-name>rest</servlet-name>
<servlet-class>...</servlet-class>
<load-on-startup>2</load-on-startup>
</servlet>

<servlet-mapping>
<servlet-name>rest</servlet-name>
<url-pattern>/api/*</url-pattern>
</servlet-mapping>
```

Patrones GOF:

Singleton: Asegurarse de que una clase tiene una sola instancia y proporcionar un único punto de acceso a ella. Se pone de manifiesto en la creación de la clase Rate donde existe una sola instancia de esta para contar todas las peticiones de los usuarios. A continuación se muestra un fragmento del código donde se ejemplifica el uso de este patrón:

```
public class Rate {

 private static Rate rate;

 public Rate() {
 ...
 }

 public static Rate getInstance(){
 if(rate == null)
 rate = new Rate();
 return rate;
 }
}
```

Observador: Se refiere al cambio de estado de uno de los objetos, y notifica este cambio a todos los dependientes. Se aprecia en la capa de servicios al usar el mecanismo de eventos ApplicationContext usado por Spring para declarar todos los objetos con sus dependencias. A continuación se muestra un fragmento del código donde se pone de manifiesto:

```
<bean id="restTemplate" class="org.springframework.web.client.RestTemplate">
 <property name="messageConverters" ref="marshallingHttpMessageConverter" />
</bean>
```

Contrato Uniforme: Es un patrón usado exclusivamente en servicios web, ya que plantea el uso de los cuatro verbos principales del protocolo HTTP en tareas específicas. Se pone de manifiesto en la capa de servicios debido al uso de estos verbos de forma estándar para las siguientes tareas:

- POST: Crear recursos.
- GET: Servicios que devuelven el estado de los recursos.
- PUT: Para actualizar recursos dado la URI del mismo
- DELETE: Eliminar un recurso y después la URI de la misma no es válida.

2.6.3. Diagrama de Clases del Diseño

Los diagramas de clases representan un conjunto de interfaces, colaboraciones y sus relaciones. Gráficamente son una colección de nodos y arcos. Muestran una serie de clases, elementos y contenidos, representados a través de las relaciones entre ellos, conformando de esta manera la estructura el sistema (Perez Cuvit, 2011). Un diagrama de clases del diseño es un tipo de diagrama estático que describe la estructura de un sistema mostrando sus clases, atributos y las relaciones entre ellos. Estas clases se especifican utilizando la sintaxis del lenguaje de programación elegido (Pressman, 2012).

A continuación se muestra algunos de los diagramas más importantes de la capa de servicios web:

- Diagrama de Clase del Diseño (DCD) Servicios del Sistema: En este diagrama están contenidos los servicios web de autenticar usuario, recuperar contraseña y generar llave de desarrollador.

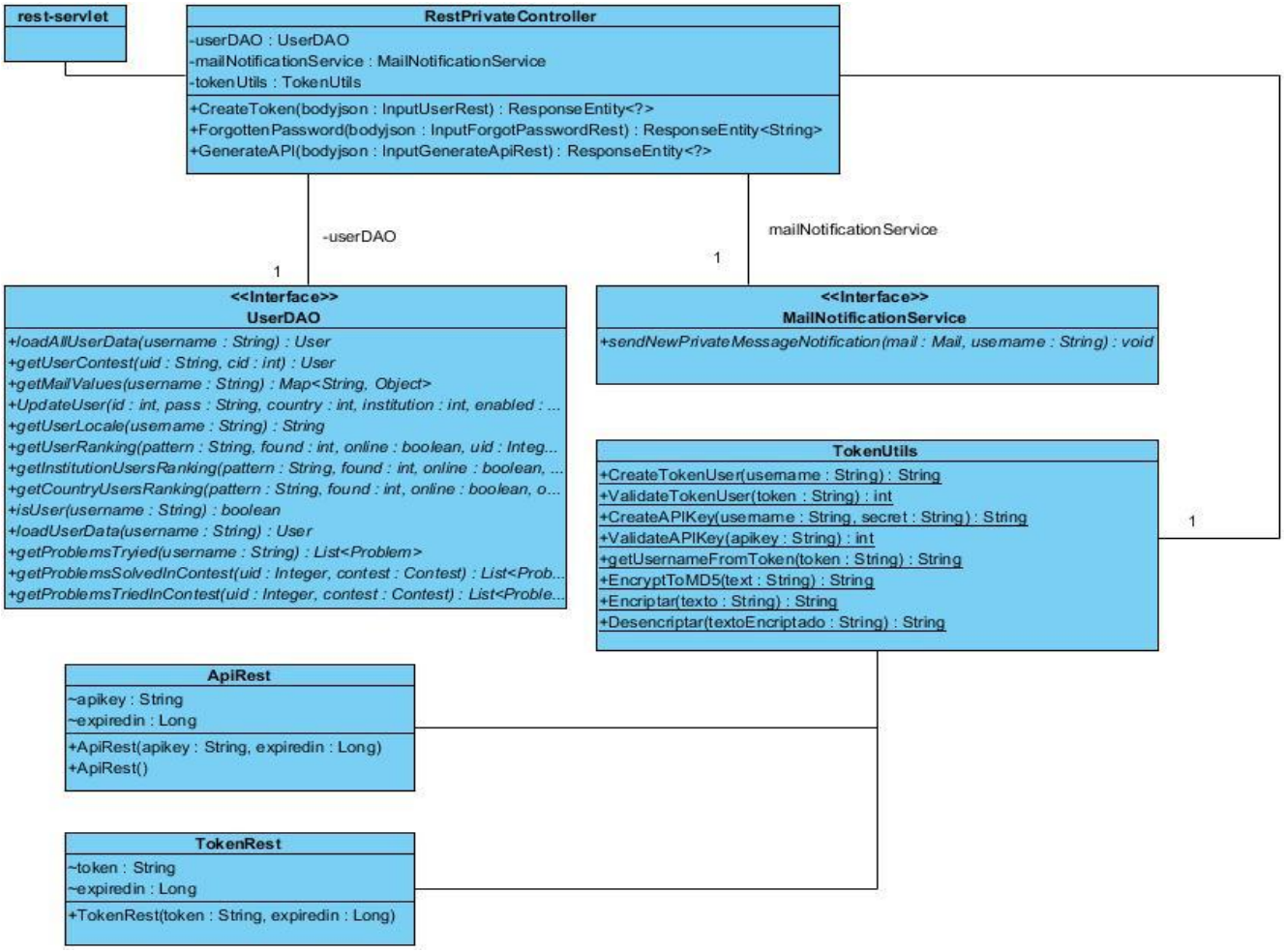


Ilustración 11 DCD Servicios del Sistema.

- DCD Servicios de Envíos: Se encuentran todos los servicios web que interactúan de alguna manera con los envíos de solución a los problemas del COJ. Entre los que se encuentran Listar todos los envíos del archivo 24 horas ya sea completos o fragmentados por páginas; Hacer un envío de solución a un problema y listar los mejores envíos de un problema.

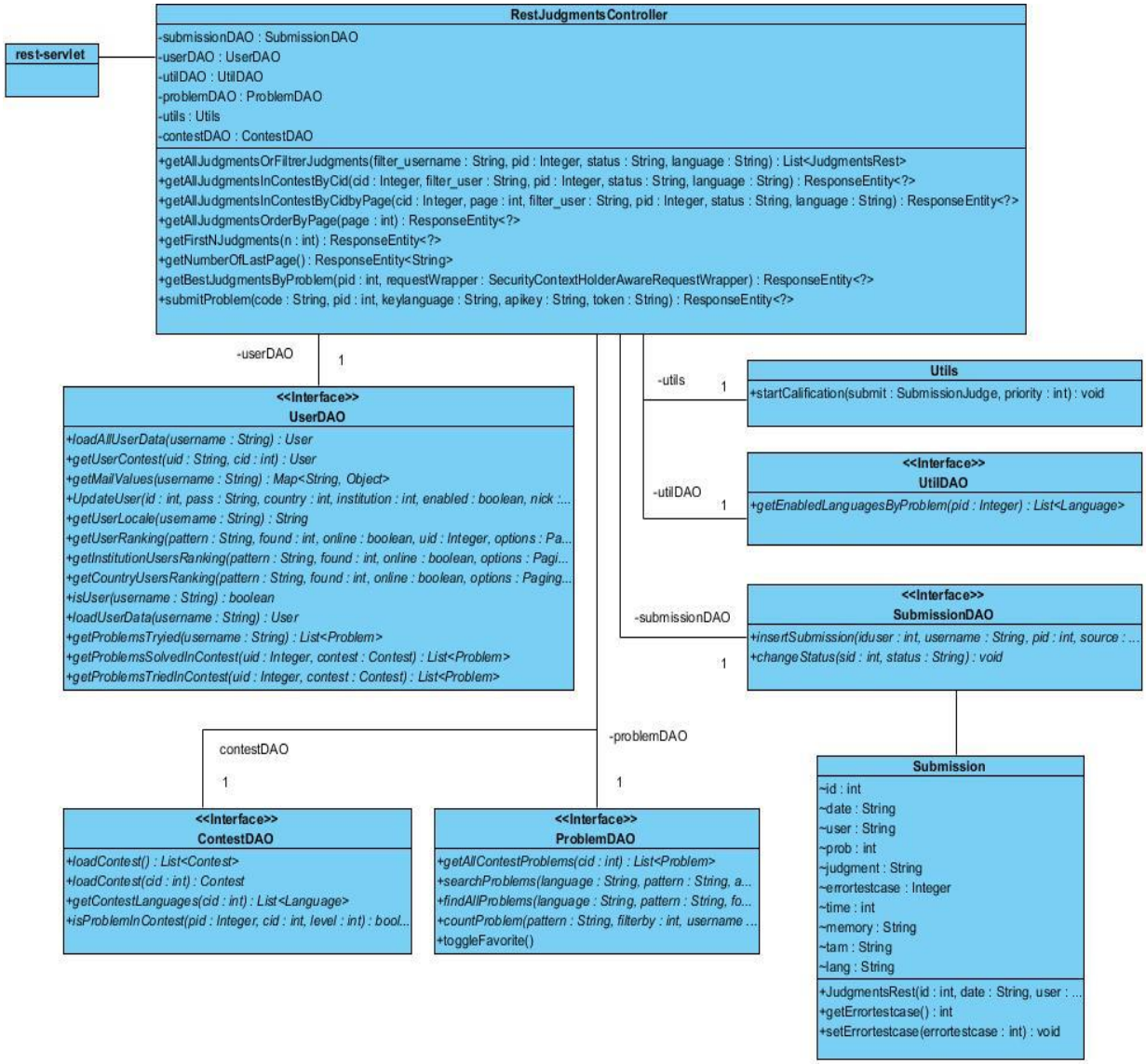


Ilustración 12 DCD Servicios de Envíos.

- DCD Servicios de problemas: Están contenidos los servicios relacionados con los problemas del COJ, entre ellos destacan listar los problemas ya sea completos o fragmentados por páginas; ver su descripción y actualizar el estado de favorito del mismo.

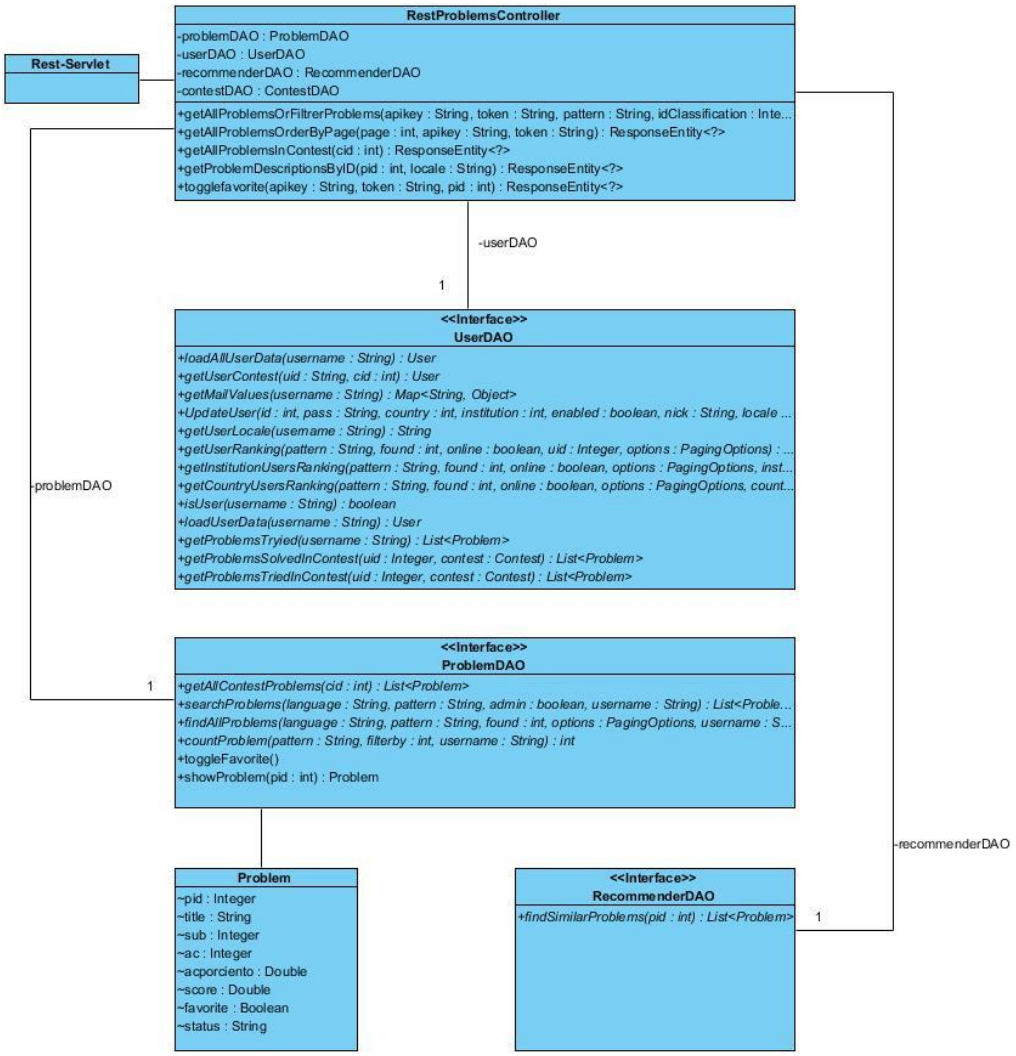


Ilustración 13 DCD Servicios de problemas.

2.6.4. Diagrama de secuencia

Un diagrama de secuencia muestra una interacción, que representa la secuencia de mensajes entre instancias de clases, componentes, subsistemas o actores. El tiempo fluye por el diagrama y muestra el flujo de control de un participante a otro. (Microsoft, 2015)

En el siguiente diagrama de secuencia se puede apreciar las interacciones entre los distintos objetos que componen el escenario Listar problemas del archivo 24 horas. La aplicación externa hace una petición al controlador frontal, luego este dirige la petición a la clase controladora RestProblemsController, esta accede

mediante el método Obtener Problemas de la clase ProblemDAO a todos los problemas de contenidos en la base de datos y es enviada una respuesta a la aplicación externa en formato JSON.

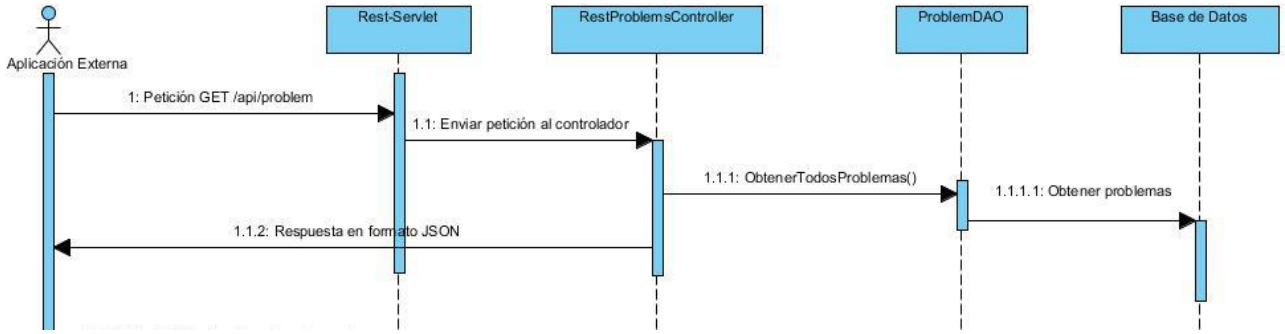


Ilustración 14 DS Listar problemas del archivo 24 horas.

Otros ejemplos de diagramas de secuencia utilizados son los siguientes:

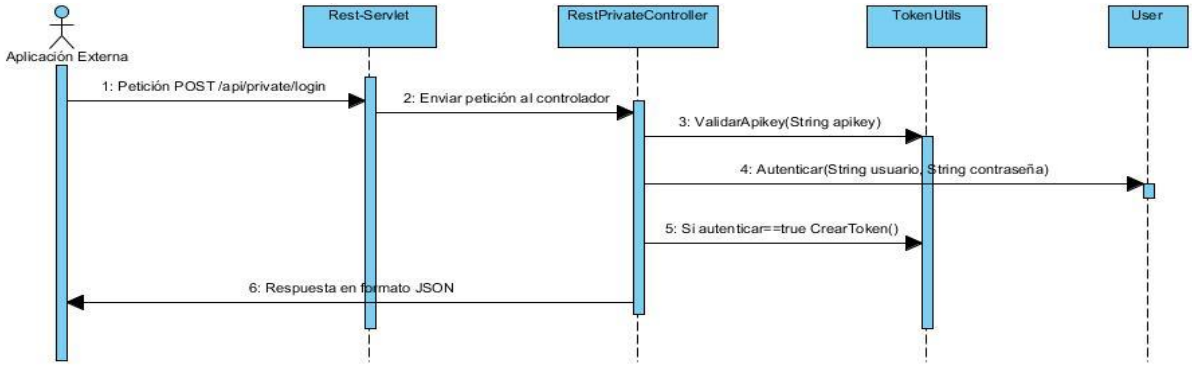


Ilustración 15 DS Autenticar Usuario.

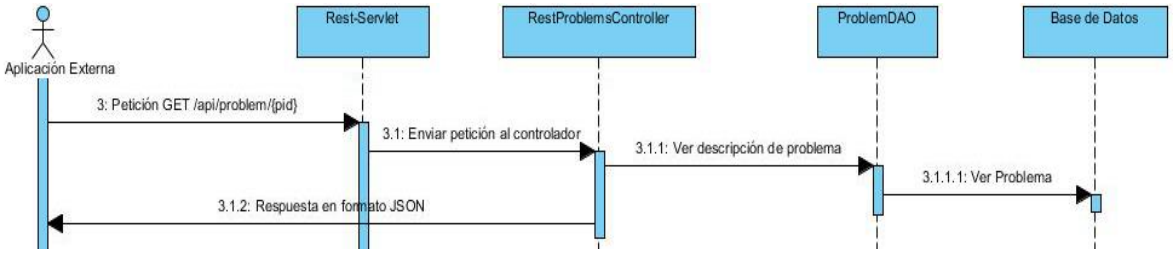


Ilustración 16 DS Ver datos de un problema.

2.6.5. Diagrama de despliegue

Un diagrama de despliegue modela la arquitectura en tiempo de ejecución de un sistema. Esto muestra la configuración de los elementos de hardware (nodos) y muestra cómo los elementos y artefactos del software se trazan en esos nodos. (SparxSystems, 2007)

El diagrama de despliegue de la capa de servicios web para el COJ que se muestra en la figura, está estructurado de la siguiente forma: el dispositivo externo se conecta al servidor web mediante el protocolo de comunicación HTTP(s) y este a su vez a la base de datos a través de TCP/IP. Además el servidor web estará conectado con el servidor de cola de mensajes (mediante TCP/IP) que se conecta a su vez con el motor de calificaciones de los envíos, usando igualmente la familia de protocolos TCP/IP.

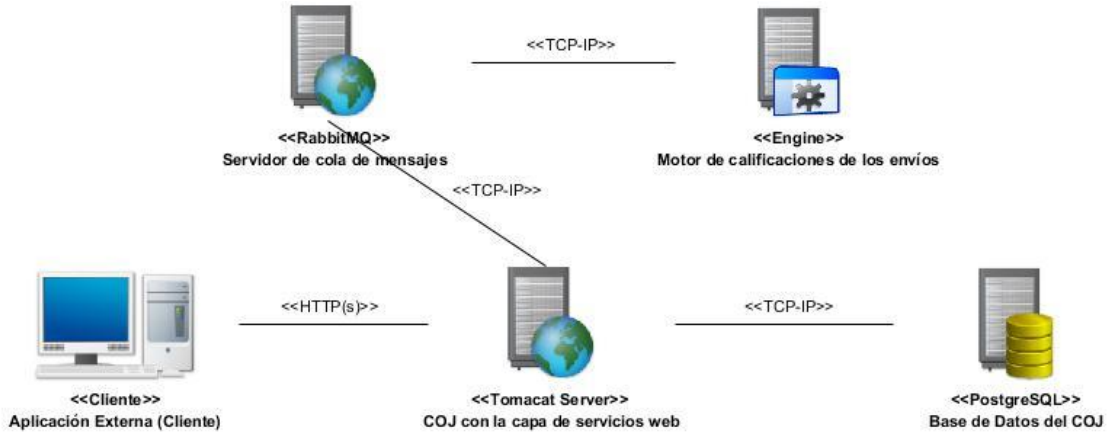


Ilustración 17 Diagrama de despliegue.

2.6.6. Diseño de la Base de datos

El Modelo de Datos es el lenguaje orientado a describir la Base de Datos, permite almacenar, organizar, manipular cantidades de datos con cierta facilidad y describir los elementos de la realidad que intervienen en el problema dado y la forma en que se relacionan estos elementos entre sí. Cuando se utiliza una base de datos para gestionar información, se está plasmando una parte del mundo real en una serie de tablas, registros y campos ubicados en un ordenador.

El sistema que se propone hace uso de la base de datos del COJ, pues la información manipulada por la capa se encuentra en esta. A continuación se muestra el modelado de la base de datos del COJ pero solo

mostrando las entidades que hace uso la capa de servicios web ya que las demás no son relevantes para la implementación de la misma.

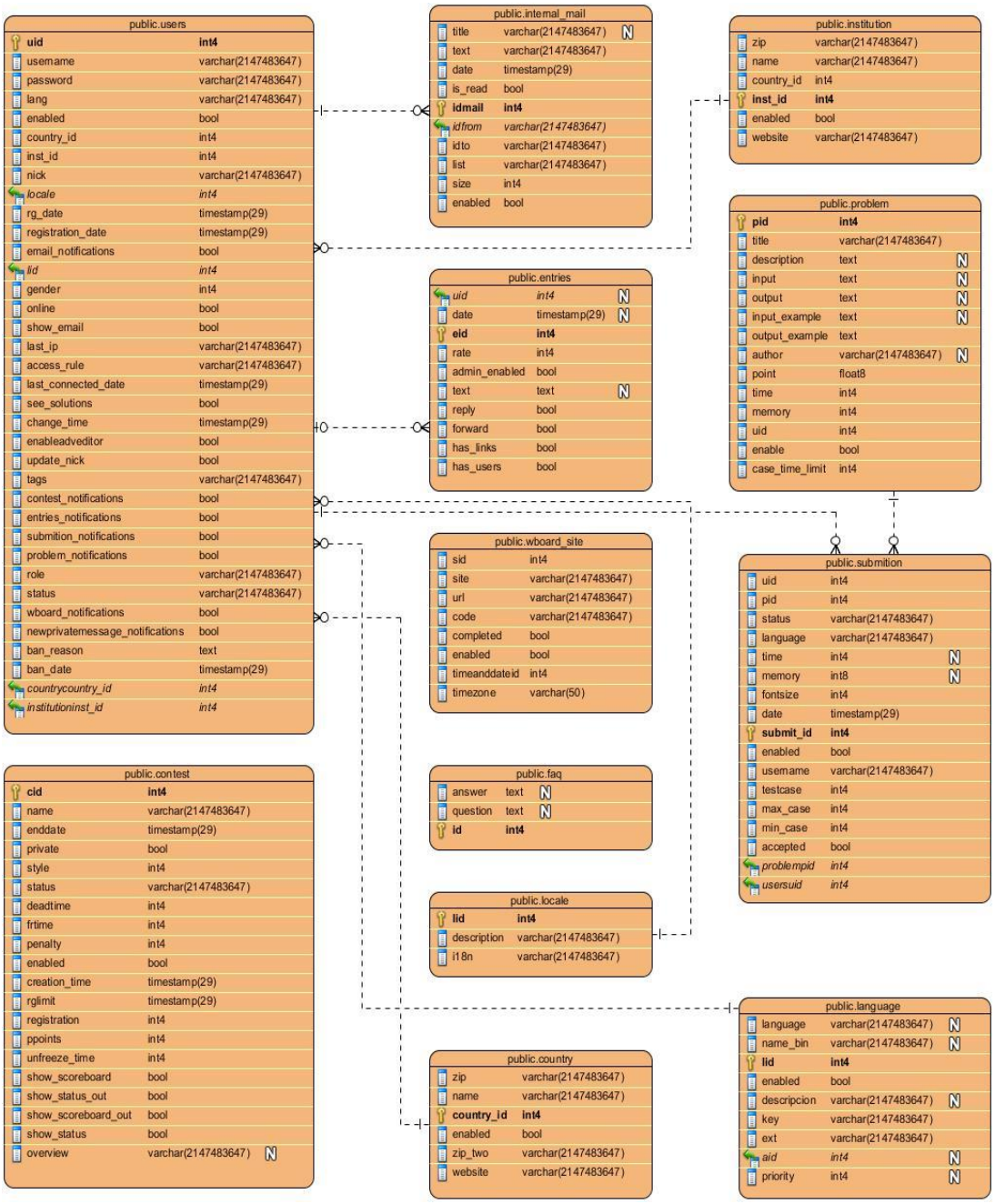


Ilustración 18 Modelo de datos.

2.7. Conclusiones parciales

- La representación del modelo del dominio sirvió de base para comprender el funcionamiento estructural entre las diferentes clases que integran el sistema. Lo que permitió establecer la vía para la identificación de los requerimientos funcionales y no funcionales que debe cumplir la aplicación.
- La modelación de historias de usuario para el encapsulamiento de los requisitos resultó ser un método eficaz para la descripción y especificación de los mismos.
- El diseño de una arquitectura n-capas y el uso de los patrones seleccionados garantizó la compatibilidad de la capa de servicios web dentro del proyecto; el encapsulamiento, dependencias, acoplamiento, asignación de responsabilidades y herencia de las clases se acopla a la arquitectura planteada por el COJ.
- De esta forma, la modelación y el diseño crearon la base para la implementación, como siguiente fase de desarrollo, que logra dar cumplimiento a los requisitos tanto funcionales como no funcionales descritos.

Capítulo 3: Implementación y Pruebas

3.1. Introducción

El período de implementación y pruebas es el momento en el que luego de haber realizado el diseño del sistema, se comienza la elaboración de los elementos de código que convertirán lo planificado en un producto real. En esta fase del desarrollo se representa la estructura física del código mediante el diagrama de componentes. Posteriormente se realizan las pruebas al software que permitan verificar la correcta implementación de los requisitos establecidos. El desarrollo de la propuesta de solución concluye con una valoración de los resultados obtenidos durante esta fase.

3.2. Diagrama de componentes

Los diagramas de componentes representan la estructura física del código. Asignan la vista lógica de las clases del proyecto a los archivos que contienen el código fuente en el que se implementa la lógica. Cuando se genera código representan la ubicación de los archivos de código fuente para sus clases. Al realizar ingeniería inversa en un proyecto ya existente pueden ayudar a establecer relaciones entre cada diagrama de clases y los archivos de código fuente. El uso más importante de este diagrama es mostrar la estructura de alto nivel del modelo de implementación, especificando los subsistemas de implementación, sus dependencias a la hora de importar el código y para organizar los subsistemas de implementación en capas y en paquetes. (ALTOVA, 2016)

Desde el punto de vista del diagrama de componentes se tienen en consideración los requisitos relacionados con la facilidad de desarrollo, la gestión del software, la reutilización, las restricciones impuestas por los lenguajes de programación y las herramientas utilizadas en el desarrollo.

El diagrama de componentes fue elaborado de forma tal que reflejara las divisiones e interdependencias entre las clases y capas de la arquitectura. Si se explica comenzando por la capa de inferior y siguiendo el proceso hasta devolver la respuesta al usuario, se puede observar como el paquete Modelo del diagrama (Ilustración 19) representa las tablas de la base de datos, a las cuales se tiene acceso solamente a través del paquete DAO. Los elementos representados en este paquete son los encargados de implementar los métodos de acceso a la base de datos para la clase a la que están destinados. Además se hace una representación del paquete que contiene las librerías de las que dependen los componentes del sistema.

En el nivel superior se refleja el núcleo de la capa de servicio web, en él están las clases controladoras que contienen todos los servicios web divididos por la tarea que realizan en el COJ y la capa Presentación que contiene un controlador frontal que desvía las peticiones que son para la capa y se la entrega a la clase correspondiente. La siguiente figura muestra el diagrama de componentes de la capa de servicios web del COJ.

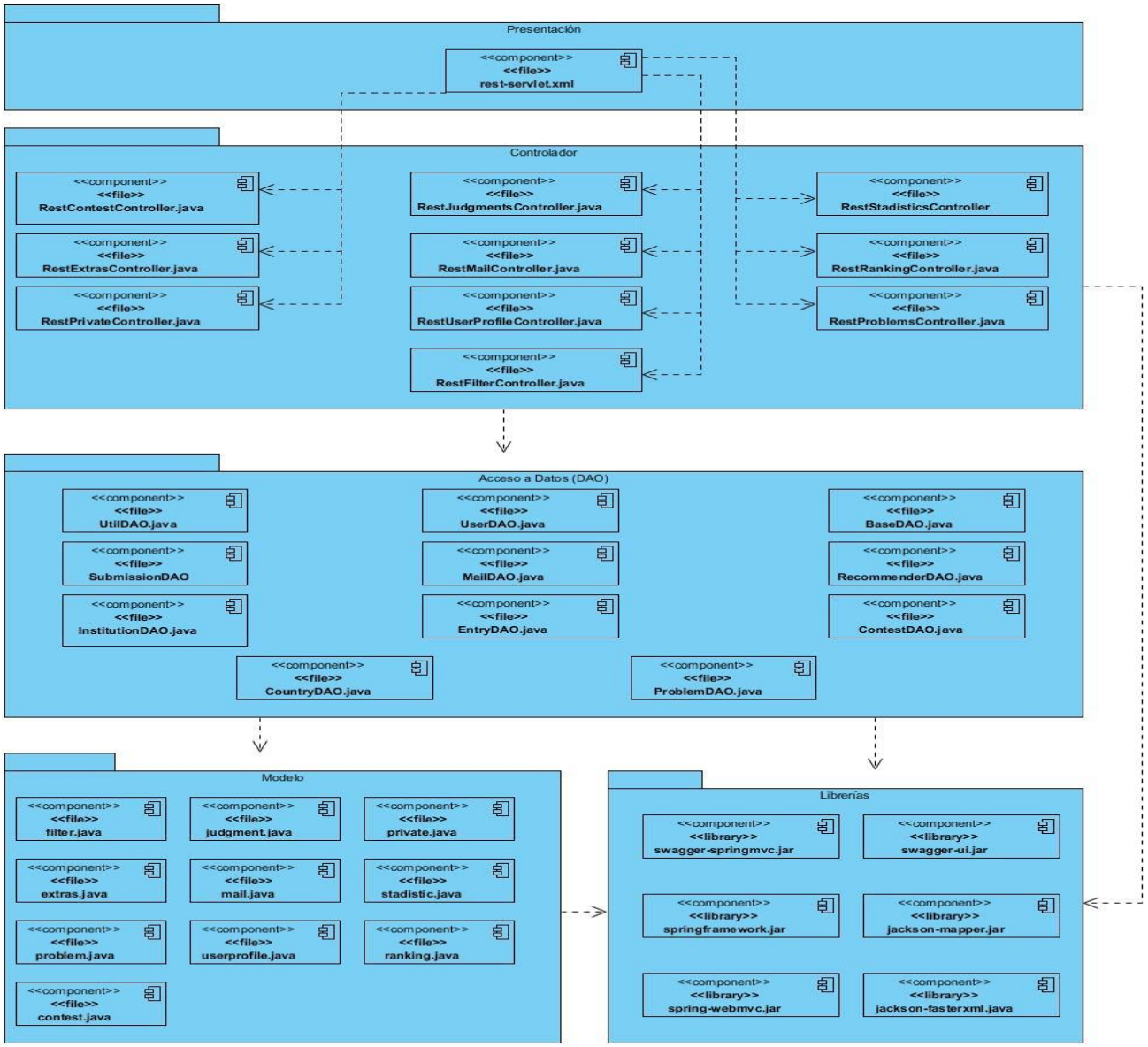


Ilustración 19 Diagrama de componentes de la capa de servicios web del COJ.

3.3. Estándares de código

Estándar de código, convención de código, o estilo de programación es un término que describe convenciones para escribir código fuente en cierto lenguaje de programación.

Las convenciones de código son importantes para los programadores debido a que el 80% del coste del código de un programa va a su mantenimiento. Casi ningún software lo mantiene toda su vida el autor original. Las convenciones de código mejoran la lectura del software, permitiendo entender código nuevo mucho más rápidamente y más a fondo. Para que funcionen las convenciones, cada persona que escribe software debe seguir la convención. (Sun Microsystems, Inc, 1999)

Para la implementación de la capa de servicios web, se seleccionó el lenguaje de programación Java. La propuesta de solución cumple con el documento “Convenciones del Código Java” establecido por la empresa *Sun Microsysytems.*

3.4. Tratamiento de errores.

Durante el desarrollo del sistema, se han definido una serie de acciones que pueden provocar un mal funcionamiento, para cada una de ellas se definió un tratamiento específico, para asegurar una respuesta adecuada que garantice cierto grado de estabilidad. El tratamiento de errores debe estar enfocado tanto a los errores que se producen a la hora de los usuarios introducir datos en la pantalla principal de la aplicación, como a los errores que puedan ser generados por el comportamiento incorrecto de los componentes internos.

En la capa de servicios web el seguimiento que se le da a los datos introducidos por los usuarios es mediante validaciones que se les asignan a los parámetros de entrada, garantizando que se muestre el tipo de error sobre el parámetro donde se introdujeron los datos incorrectos. Una ventaja de darle seguimiento a los errores es poder señalar y separar el tratamiento de los errores mediante excepciones y poder cumplir una respuesta a errores particulares. A continuación se muestra una tabla que evidencia el tratamiento de errores. Los mensajes de error llegarán a los usuarios formato JSON, especificando el error HTTP correspondiente y un mensaje de la razón del mismo con el objetivo de facilitar su comprensión, pero sin revelar información propia del sistema que pueda atentar contra la seguridad. A continuación se muestra un ejemplo de un mensaje de error:

Ilustración 20 Ejemplo de mensajes de error en una respuesta de la capa de servicios web.

En la siguiente tabla se describen los posibles errores HTTP y la razón del mismo:

# error HTTP	Error	Razón
400	Petición Incorrecta.	Incluye una serie de errores relacionado con las peticiones que hacen los clientes hacia la capa de servicio si no cumplen estas con las especificaciones requeridas.
401	No autorizado.	Incluye una serie de errores que indican que no están autorizados los clientes a ejecutar esa petición en el sistema. Ya sea porque no están autentificados o expiró el tiempo del token.
404	No encontrado	Incluye una serie de errores en las peticiones de los clientes a la hora de pedir un recurso que no existe en el sistema.
412	Pre-condición fallida	Ocurre cuando algún parámetro es enviado correctamente pero no cumple con la(s) condición(es) para pasar a la ejecución del servicio. Ejemplo: textos muy largos o demasiados cortos, etc.
429	Excede el límite de peticiones	Cuando se intenta hacer demasiadas peticiones a un servicio web.

Tabla 8 Posibles errores que contienen las respuestas de los servicios.

3.5. Pantalla principal de la aplicación

Ilustración 21 Pantalla principal de la capa de servicios web.

3.6. Aspectos de seguridad en el sistema.

Teniendo en cuenta que el medio de publicación (Internet) de la capa de servicios web y que cualquier persona con acceso a este medio puede acceder a la capa de servicios web, se tomaron una serie de medidas que ayudarán a garantizar la seguridad del sistema que incluye la confiabilidad, integridad y disponibilidad de los datos.

Se usó un sistema de autentificación basado en tokens, donde el mismo es creado y enviado al cliente cuando se autentifica un usuario en el sistema mediante la capa, y se utiliza cuando se desea consumir algún servicio que la información a mostrar del mismo solo sea visible para ese usuario. Para ello, previamente el desarrollador tiene que poseer una llave que le da acceso a los métodos que requieren autentificación. Ambos, el token y la llave de desarrollador, van a ser únicos y contarán con una encriptación

combinada de función hash, más base64 que se generará a partir del usuario y contraseña enviado, más una clave ya predefinida en el COJ. El token tendrá una vigencia de quince minutos a partir de que se autentifique el usuario, después de esto, solo se comprobará su identidad a través de dicho token, evitando que datos sensibles como el usuario y la contraseña viajen por el canal de comunicación frecuentemente.

Por otra parte, se ha tenido en cuenta utilizar HTTPS, que es la versión segura del protocolo HTTP y viene dada por el protocolo de seguridad SSL (del inglés: *Secure Sockets Layer*). Además se limitó el número de peticiones que se le pueden hacer a ciertos servicios de alta demanda y al servidor en general para evitar ataques de denegación de servicios o ataques de fuerza bruta al servicio de autentificación de usuarios.

3.7. Modelo de prueba

Para que la capa de servicios web se considere lista para ser desplegada y en funcionamiento en el COJ, debe someterse previamente a una etapa de pruebas rigurosas, con el fin de analizar si cumple con las funcionalidades requeridas. Las pruebas que se le realizarán a la aplicación son un elemento crítico para la garantía de la calidad, representan además la revisión final de las especificaciones, los requerimientos, el análisis, diseño e implementación del módulo. El principal objetivo de estas pruebas es descubrir errores; estos serán corregidos posteriormente para que la aplicación final cumpla con lo previsto y tenga la eficiencia y la calidad requerida. Aunque las pruebas no pueden asegurar la ausencia de defectos; sí pueden demostrar la existencia de estos (Pressman, 2012).

El modelo de prueba describe principalmente cómo se prueban los componentes ejecutables en el modelo de implementación con pruebas de integración y de sistema. El modelo de pruebas puede describir también cómo han de ser probados aspectos específicos de sistemas, por ejemplo, si el manual de usuario del sistema cumple con su cometido. El modelo de pruebas es una colección de casos de prueba, procedimientos de prueba y componentes de prueba (Vivv, 2008).

Los requisitos mínimos de hardware y software son necesarios para la realización exitosa de cualquiera de las pruebas. Se realizó un análisis de los requerimientos del sistema y su disponibilidad para ser usado, así como las herramientas de software necesarias para la realización de las pruebas.

Descripción de los tipos de prueba realizadas

Para la realización de las pruebas al sistema, se definieron una serie de pruebas fundamentales para la validación de los servicios web:

Pruebas Funcionales: Tiene como objetivo asegurar el cumplimiento apropiado de los requisitos funcionales, entrada de datos, procesamiento y obtención de resultados. La principal intención de estas pruebas es medir la correspondencia entre la arquitectura de información propuesta y las funciones que realmente fueron implementadas. Para realizar este tipo de pruebas es necesario tener definidos los requerimientos a verificar con los casos de prueba apuntando a cada uno de ellos. Estos serán los encargados de verificar la aplicación implementada. Se utilizará el método de caja negra.

Pruebas de Rendimiento: La prueba de rendimiento de software se enfoca en la capacidad de recibir peticiones mediante la utilización de alguna herramienta, verificando cuantas peticiones pueda sostener el sistema sin que este se vea afectado, así como la velocidad de respuesta del mismo. Las pruebas de rendimiento para la capa de servicios web serán realizadas utilizando la herramienta JMeter, que es una aplicación de escritorio y de código abierto, diseñada para realizar pruebas de carga y stress, medir los tiempos de las pruebas y medir el rendimiento de un sistema. JMeter también puede ser utilizado como una herramienta de prueba de carga para analizar y medir el desempeño de una variedad de servicios, con énfasis en aplicaciones web.

Pruebas de Seguridad: La prueba de seguridad y control de Acceso se enfocará a la seguridad en el ámbito de la aplicación, asegurando que los que los mecanismos de protección incorporados en el sistema lo protegerán de accesos impropios y que los usuarios solo accedan a los servicios que requieran autentificación cuando estén debidamente registrados y autentificados en el sistema mediante la capa de servicios.

3.7.1. Pruebas funcionales

Las pruebas se le realizaron a cada uno de los requisitos de la capa de servicios web. En el desarrollo de la capa se diseñaron un conjunto de casos de prueba de las diferentes funcionalidades del sistema. A continuación, se describen algunos de los casos de prueba utilizados.

Historia de Usuario	HU_10 Listar problemas
Requisito	RF10
Condición	Debe existir un listado de problemas en el archivo de 24 horas.

Tabla 9 Caso de prueba: Listar problemas.

No	Nombre del campo	Clasificación	Valor Nulo	Descripción
1	cid	Campo de texto	No	Puede contener solo números
2	page	Campo de texto	No	Puede contener solo números

Tabla 10 Descripción de variables. Caso de prueba: Listar problemas.

Escenario	Descripción	cid	page	Respuesta del sistema	Flujo central
EC 1.1 Listar problema	Devuelve un listado de todos los problemas que existen en el archivo 24 horas.	N/A	N/A	Devuelve todos los problemas existentes en el archivo 24 horas. Se muestran los siguientes datos: Identificador del problema, título, envíos, aceptados, puntuación.	/problem
EC 1.2 Listar problema por páginas	Devuelve un listado de todos los problemas que existen en el archivo 24 horas paginados.	N/A	V	Devuelve todos los problemas existentes en el archivo 24 horas. Se muestran los siguientes datos: Identificador del problema, título, envíos, aceptados, puntuación.	/problem/page/{page]
EC 1.3 Listar problema por competencia	Devuelve un listado de todos los problemas	V	N/A	Devuelve todos los problemas existentes en una determinada competencia. Se muestran los siguientes datos: Identificador	/problem/contest/{cid}

	que existen en una determinada competencia			del problema, título, envíos, aceptados, puntuación.	

Tabla 11 Diseño de caso de prueba: Listar problemas.

Historia de Usuario	HU_9 Autenticar usuario
Requisito	RF9
Condición	Debe existir el usuario registrado en el sistema.

Tabla 12 Caso de prueba: Autenticar usuario.

No	Nombre del campo	Clasificación	Valor Nulo	Descripción
1	apikey	Campo de texto	No	Admite cualquier carácter.
2	usuario	Campo de texto	No	Admite cualquier carácter.
3	contraseña	Campo de texto	No	Admite cualquier carácter.

Tabla 13 Descripción de variables. Caso de prueba: Autenticar usuario.

Escenario	Descripción	Usuario	Contraseña	Apikey	Respuesta del sistema	Flujo central
EC 1.1 Autenticar usuario.	Autentica al usuario y devuelve un token de usuario.	V	V	V	Devuelve todos los problemas existentes en el archivo 24 horas. Se muestran los siguientes datos: Identificador del problema, título, envíos, aceptados, puntuación.	/problem
EC 1.2 Datos incorrec-tos	Datos incorrectos.	I	V	V	Devuelve error HTTP 401	/problem/page/{page}
		V	I	V		
		V	V	I		

Tabla 14 Diseño de casos de prueba: Autenticar usuario.

Resultados de las pruebas funcionales

La implementación de las pruebas funcionales muestra una visión de la calidad del software. En una primera iteración, se obtuvo como resultado del análisis de los escenarios de pruebas, un conjunto de no conformidades correspondientes a los servicios web implementados. Las no conformidades fueron corregidas posteriormente, dándole paso a la segunda iteración de pruebas, en las cuales se evaluaron nuevamente, obteniendo otro conjunto de nuevas de no conformidades y así sucesivamente hasta un total de cuatro iteraciones. A continuación se muestran los principales datos de cada iteración:

Iteración	No conformidades	Descripción (no conformidades más significativas)
1	12	• El formato de los errores en las respuestas no estaban en JSON. • Respuestas incorrectas de algunas funcionalidades como listar correos y actualizar estado de favorito de los problemas. • Error al enviar correos después de cambiada la contraseña mediante la capa de servicios web.
2	9	• Errores a la hora de visualizar la documentación de los servicios generada por la librería OpenAPI. • Algunos servicios no cumplían con los estándares REST a la hora de consumirlos.
3	3	• Los identificadores de errores HTTP no coincidían con los expuestos en la documentación. • Errores ortográficos en la descripción de los servicios.
4	0	

Tabla 15 No conformidades por iteración.

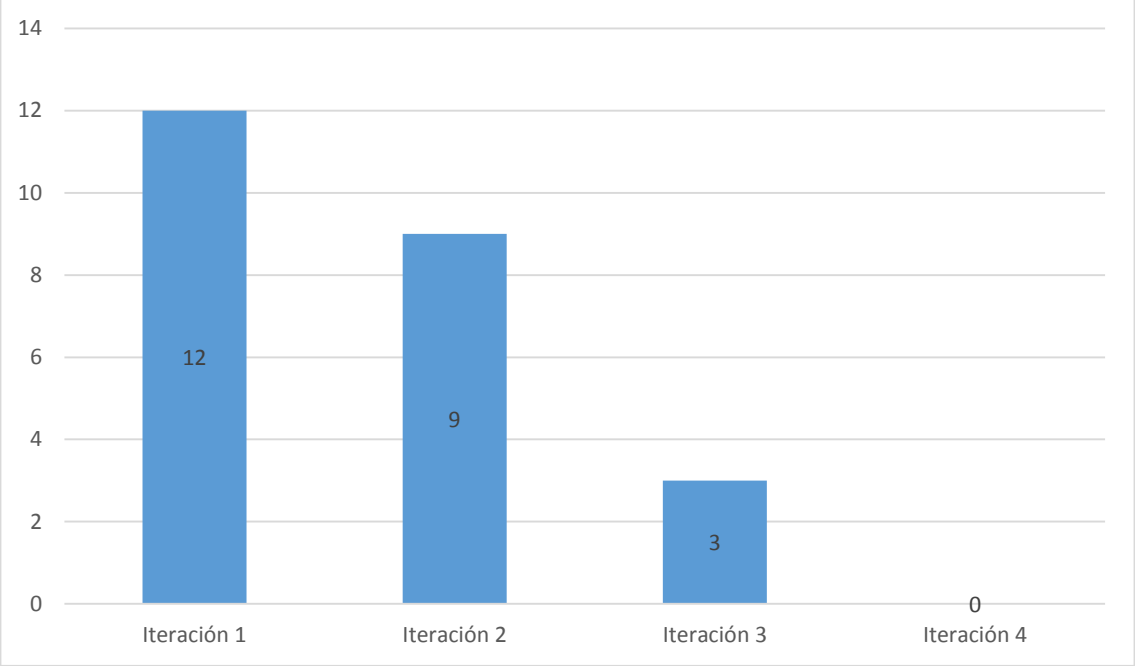


Ilustración 22 Resultados de las pruebas funcionales.

Se validó en sentido general el correcto funcionamiento y cumplimiento de los requisitos funcionales de la capa de servicios, de esta forma el desarrollo de las pruebas funcionales muestra una visión de la calidad del software. Para una total verificación de las funcionalidades de la capa de servicios web se utilizó la interfaz de prueba generada con el OpenAPI para probar cada una de ellas. Esta aplicación confirmó el correcto funcionamiento de la capa y también servirá de apoyo como documentación para los usuarios interesados en utilizar la capa de servicios web.

3.7.2. Pruebas de rendimiento

Para probar la escalabilidad del sistema es necesario conocer acerca de la carga que puede soportar. Para ello se implementaron las pruebas de rendimiento de software, las cuales se apoyaron en el uso de la herramienta JMeter, que ofrece datos importantes acerca del estrés que puede tolerar el sistema.

Para la realización de las pruebas se utilizó el entorno de hardware y software del servidor oficial del COJ que el mismo consiste en:

- PC con procesador i7 a 3.0 GHZ y 16 GB de RAM.
- Sistema operativo Ubuntu 10.04 en su versión para servidores.

- Entorno de ejecución java (JRE) en su versión 1.7.
- Servidor Web Apache Tomcat en su versión 7.0.
- Servidor de base de datos.
- PostgreSQL en su versión 8.4.

Resultado de las pruebas de rendimiento.

Se probó el sistema con un total de **2200** peticiones enviadas en **100** hilos de ejecución concurrentes en once servicios web, arrojando un **2.59%** de error total al atender dichas peticiones a los servicios. El porcentaje de error estuvo mayormente originado por paquetes perdidos, debido a que la prueba se realizó a través de una red inalámbrica y al firewall del servidor del COJ bloquea ciertas peticiones por el abuso de envíos. No fueron responsabilidad de la capa de servicios.

Label	# Muestras	Media	Mín	Máx	Std. Dev.
Listar problemas	200	18744	7596	33523	5731.12
Listar problemas paginados	200	981	28	7127	1263.88
Listar envíos	200	10710	1874	20448	3874.18
Listar envíos paginados	200	1356	331	9335	1145.02
Ver perfil de usuario	200	6296	525	15757	3133.86
Estadisticas 24 horas	200	438	6	7913	802.95
Próximas competencias	200	501	6	7453	868.27
Competencias pasadas	200	3019	62	11406	2372.92
Competencias Actuales	200	327	0	4072	597.84
Listar lenguajes disponibles	200	355	0	4511	634.08
Comentarios	200	455	54	3768	639.65
TOTAL	2200	3925	0	33523	6162.53

Label	# Muestras	% Error	Rendimiento	Kb/sec	Avg. Bytes
Listar problemas	200	11.50%	2.4/sec	587.69	255709.6
Listar problemas paginados	200	0.50%	2.6/sec	15.05	5910.4
Listar envíos	200	8.50%	2.5/sec	202.84	83631.8
Listar envíos paginados	200	1.50%	2.7/sec	9.63	3715.1
Ver perfil de usuario	200	4.00%	2.7/sec	2.34	901.2
Estadisticas 24 horas	200	0.50%	2.9/sec	4.68	1650.8
Próximas competencias	200	0.50%	3.0/sec	1.85	641.8
Competencias pasadas	200	1.50%	3.0/sec	206.16	70301.0
Competencias Actuales	200	0.00%	3.4/sec	0.01	2.0
Listar lenguajes disponibles	200	0.00%	3.4/sec	3.80	1154.0
Comentarios	200	0.00%	3.4/sec	10.83	3295.0
TOTAL	2200	2.59%	23.7/sec	899.39	38810.3

Tabla 16 Resultados de las pruebas con JMeter.

El significado de algunos campos se describe a continuación:

Label: el nombre de la petición HTTP(s).

 # Muestras: el número de muestras para cada petición.

Media: el tiempo medio transcurrido para un conjunto de resultados.

Mínimo: el mínimo tiempo transcurrido para las muestras de la petición dada.

Máximo: el máximo tiempo transcurrido para las muestras de la petición dada.

% Error: porcentaje de las peticiones con errores.

Rendimiento: rendimiento medido en peticiones por segundo/minuto/hora.

Kb/segundos: rendimiento medido en kilobytes por segundo.

En resumen se puede decir que la capa de servicios web cumple con el rendimiento requerido (**23.7** peticiones por segundo según la herramienta) para atender la carga y stress que se genere al atender las solicitudes de los diferentes clientes que puedan conectarse a la capa.

3.7.3. Pruebas de Seguridad

Se probó manualmente un token de usuario dado por el servicio de autenticar usuario mediante la API generada con la herramienta OpenAPI y se verificó que las funcionalidades o datos adicionales están correctamente disponibles o sean denegados en caso de que sean incorrectos. Además se probó que el token solo fuera válido en el tiempo asignado y expirará después de cumplido el mismo.

Se probó que el servicio de autentificación (coj.uci.cu/api/private/login) cumpliera con la restricción de envíos por minuto que se puede realizar. Solo debe admitir treinta envíos por minutos que reciba desde un cliente y después devolver el error HTTP 429 (Demasiados envíos realizados) para así evitar ataques de fuerza bruta a los usuarios o contraseñas guardadas en la base de datos.

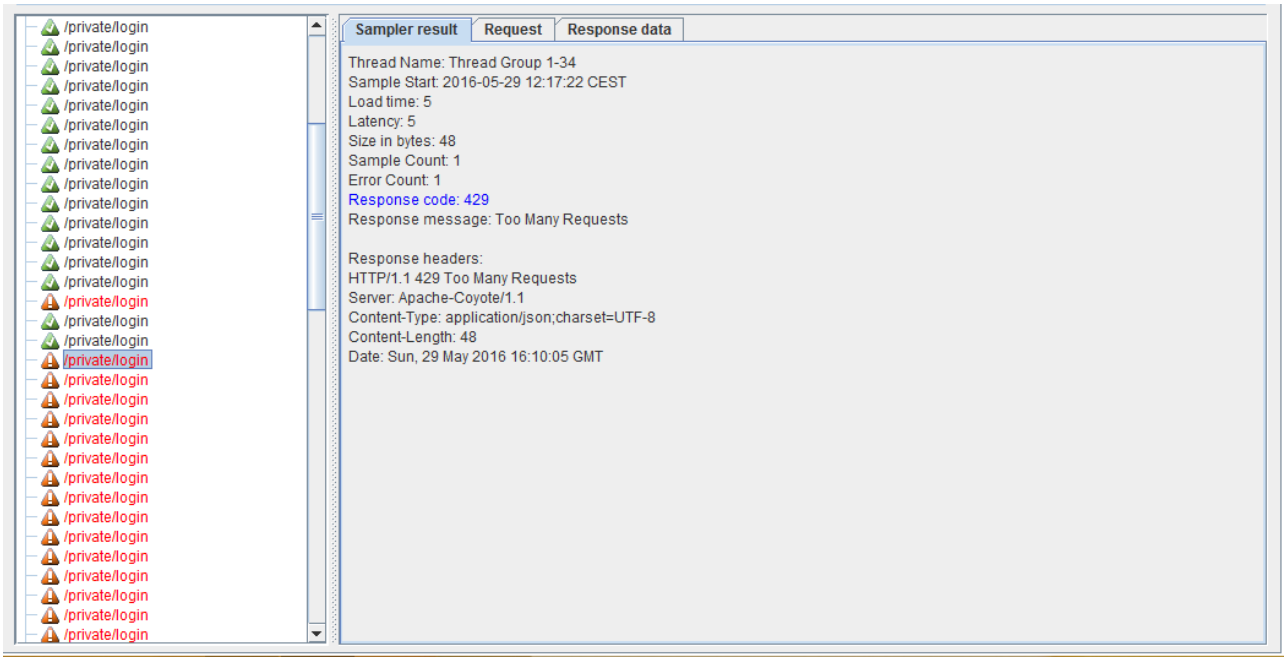


Ilustración 23 Respuesta de 100 envíos realizadas al servicio de autentificación.

A través de la aplicación *COJ Mobile* y el proyecto *PMOOCs*, ambos desarrollados en el Centro FORTES de la Universidad de las Ciencias Informáticas, se pudo demostrar la interoperabilidad de las aplicaciones con el COJ ya que estas utilizan las funcionalidades del juez a través de la capa de servicios creada, corroborando así el correcto funcionamiento de la misma.

3.8. Conclusiones parciales

- Mediante el diseño del diagrama de componente se representó las relaciones de dependencia entre los componentes del sistema, así como las librerías que utiliza, para una mejor comprensión del funcionamiento de la capa de servicios y las tecnologías que la componen.
- El tratamiento de errores del sistema se concentró en aquellos bug que se pudieran producir a la hora de los usuarios introducir datos en la pantalla principal (“sandbox”) de la aplicación, y los que puedan ser generados por el comportamiento incorrecto de los componentes internos. Por lo que se decidió darle un seguimiento a los datos introducidos por los usuarios mediante validaciones que se les asignan a los parámetros de entrada. Lo que permitió señalar y separar dicho tratamiento mediante excepciones para poder dar una respuesta a errores específicos.

- Se decidió tener en cuenta algunos aspectos de la seguridad del sistema dado que el medio de publicación de la capa de servicios web es a través de Internet, además, cualquier persona con acceso a este, podría acceder a la capa de servicios web. Por lo que se tomó una serie de medidas para garantizar la seguridad del sistema, entre las que se encuentran: usar un sistema de autentificación basado en tokens y utilizar HTTPS, que es la versión segura del protocolo HTTP y respaldada por el protocolo de seguridad SSL.
- A través del modelo de prueba se seleccionó las pruebas funcionales, de rendimiento y seguridad, se utilizó la herramienta JMeter en las pruebas de rendimiento y la validación de la aplicación mediante pruebas de caja negra. Se obtuvieron resultados satisfactorios y se demostró el correcto desarrollo de la capa de servicios web.

Conclusiones

- La sistematización del estudio del estado del arte de las herramientas y tecnologías para el desarrollo de servicios web en jurados en línea permitió implementar una aplicación actualizada tecnológicamente.
- El análisis de las diferentes metodologías y herramientas disponibles, posibilitó una adecuada selección de la estructura tecnológica usada en el desarrollo de la capa de servicios.
- El diseño de la propuesta de solución permitió la creación de la base para la implementación como siguiente fase de desarrollo, lográndose dar cumplimiento a los requisitos tanto funcionales como no funcionales descritos.
- A través de las diferentes pruebas de software ejecutadas en la capa de servicios web se validó el correcto funcionamiento del sistema, permitiendo demostrar la interoperabilidad de una aplicación informática con el Juez en Línea Caribeño. Cumpliéndose de esta forma el objetivo principal de la investigación.
- La capa de servicios web para el COJ es una ventajosa herramienta que persigue un resultado aplicable a las necesidades del proyecto. Por lo que su elaboración aporta un valor agregado al juez en línea, sentando las bases para extender su alcance y visibilidad a través del número considerable de servicios implementados, permitiendo el uso de sus funcionalidades desde cualquier aplicación de terceros desarrollada.

Recomendaciones

Con el objetivo de mejorar y seguir perfeccionando la capa de servicios, los autores del presente trabajo de diploma recomiendan al equipo de desarrollo del COJ:

- La implementación de nuevas funcionalidades a la capa de servicios web.
- Gestionar los usuarios permitidos para usar la capa de servicios web.

Referencias

Acero Martín, Fernando. 2011. SeguiInfo. [En línea] 15 de 5 de 2011. [Citado el: 20 de 1 de 2016.] https://seguinfo.wordpress.com/2009/05/15/interoperabilidad-i-las-deffiniciones/.

Akhtar, Md Mobin y Gupta, Pankaj Kumar. 2015. *A Web Services Created Online Training and Assessment Scheme.* s.l. : INPRESSCO, 2015. E-ISSN 2277 – 4106.

Alhir, Sinan Si . 2003. *Learning UML.* s.l. : O'Reilly, 2003.

Allen, P. 2007. The Service Oriented Process. CBDi Journal. [En línea] febrero de 2007. http://www.cbdiforum.com/secure/interact/2007-02/.

ALTOVA. 2016. ALTOVA. [En línea] 2016. [Citado el: 21 de abril de 2016.] http://www.altova.com/es/umodel/uml-component-diagrams.html.

Alvarez, Miguel Angel, y otros. 2014. Desarrollo Web. [En línea] Apache, 19 de 12 de 2014. [Citado el: 3 de 5 de 2016.] http://www.desarrolloweb.com/articulos/ventajas-inconvenientes-apirest-desarrollo.html.

Apache JMeter. 2016. Apache JMeter. *http://jmeter.apache.org.* [En línea] 2016. [Citado el: 22 de 3 de 2016.] http://jmeter.apache.org.

Araojo, Erik. 2016. CodePlex. *http://storm.codeplex.com/.* [En línea] 2016. [Citado el: 22 de 3 de 2016.] http://storm.codeplex.com/.

Beck, Kent. 1999. *Extreme Programming Explained.* 1999. ISBN-13: 978-0201616415.

Boci, Lin, y otros. Agosto 2012. *Comparison between JSON and XML in Applications on AJAX.* CSSS 2012 : s.n., Agosto 2012.

Booth, David y Kevin Liu, Canyang. 2011. Web Services Description Language (WSDL) Version 2.0. *W3C.* [En línea] 26 de 7 de 2011. [Citado el: 5 de 2 de 2016.] https://www.w3.org/TR/wsdl20-primer/#Introduction.

Codeforces. 2016. Codeforces. [En línea] 2016. [Citado el: 5 de 5 de 2016.] http://codeforces.com/api/help.

Craig, Larman. 2003. *UML y Patrones. 2a Edición.* 2003.

Duarte, Ailin Orjuela y Rojas C, Mauricio. 2008. *Las Metodologías de Desarrollo Ágil como una Oportunidad para la Ingeniería del Software Educativo.* Colombia : s.n., 2008. 1657-7663.

ECMA. 2013. *Standard ECMA-404 The JSON Data Interchange Format.* 2013.

Europa PRESS. 2012. *Europa PRESS.* [En línea] 20 de 4 de 2012. [Citado el: 3 de 2 de 2015.] http://www.europapress.es/castilla-y-leon/innova-00439/noticia-innova-juez-online-uva-alcanza-diez-millones-accesos-procedentes-todo-mundo-20120420142158.html.

Expósito, Erly Delgado. 2008. *Metodologías de desarrollo de software. ¿Cuál es el camino?* Matanzas, Cuba : s.n., 2008. 1990-8830.

Fielding, Roy T. 2000. Architectural Styles and the Design of Network-based Software Architecture. *PhD Thesis.* [En línea] 2000. [Citado el: 4 de 2 de 2016.] http://roy.gbiv.com/pubs/dissertation/top.htm.

Fowler , Martin . 2010. Martin Fowler. [En línea] 18 de 3 de 2010. [Citado el: 5 de 5 de 2016.] http://martinfowler.com/articles/richardsonMaturityModel.html.

Francia, Steve. 2010. spf13-REST vs SOAP, the difference between soap and rest. *spf13.* [En línea] 15 de 1 de 2010. [Citado el: 5 de 2 de 2016.] http://spf13.com/post/soap-vs-rest.

González Quiroga, María. 2011. *Estudio de Arquitecturas de Redes Orientadas a Servicio.* Barcelona : Universidad de Catalunya, 2011.

Google. 2016. Google Trends. [En línea] 2016. [Citado el: 5 de 5 de 2016.] http://www.google.com/trends/explore?q=xml+api#q=xml%20api%2C%20json%20api&cmpt=q.

Gosling, James , y otros. 2015. *The Java Language Specification-Java SE 8 Edition.* Redwood : Oracle America, Inc., 2015. Vol. 8.

Hadley, Marc. 2009. Web Application Description Language. *W3C.* [En línea] 31 de 8 de 2009. [Citado el: 5 de 2 de 2016.] https://www.w3.org/Submission/wadl/.

Hussain, Shariq, y otros. 2013. *Web Service Testing Tools: A Comparative Study.* Beijing, China : School of Computer and Communication Engineering, University of Science and Technology Beijing, 2013.

IBM. 2007. Rational Unified Process. [En línea] 2007. http://www.ibm.com.

IDABC. 2009. *EUROPEAN INTEROPERABILITY FRAMEWORK FOR PAN-EUROPEAN eGOVERNMENT SERVICES.* s.l. : European Communities, 2009. ISBN 92-894-8389-X.

IEEE. 1990. *IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries.* New York, NY : s.n., 1990.

Johnson, Rod, y otros. 2015. Spring Framework Reference Documentation. *spring.io.* [En línea] 2015. [Citado el: 9 de 2 de 2016.] http://docs.spring.io/spring/docs/current/spring-framework-reference/html/.

Junco Vázquez, MSc. Tomás Orlando. 2012. *UN JUEZ EN LÍNEA AJUSTADO A LAS NECESIDADES DE LA DOCENCIA.* La Habana : s.n., 2012.

Klensin, J. 2010. FILE TRANSFER PROTOCOL (FTP). *FILE TRANSFER PROTOCOL (FTP).* [En línea] 3 de 2010. [Citado el: 5 de 2 de 2016.] https://tools.ietf.org/html/rfc5797.

Larman, Craig. 2001. UML y Patrones - Una introducción al análisis y diseño orientado a objetos y al proceso unificado. s.l. : Prentice Hall, 2001. Vol. 2.

Linux Foundation Collaborative Project. 2016. OpenAPI. [En línea] 2016. [Citado el: 2 de 5 de 2016.] https://openapis.org/.

Macas, Ángel Miguel. 2009. Monografías. *http://www.monografias.com.* [En línea] 20 de 8 de 2009. [Citado el: 21 de 3 de 2016.] http://www.monografias.com/trabajos73/diseno-software/diseno-software2.shtml.

María Gabriela Farías Terrens, Andrea Infante Salgado. *Seguridad Informática en Web Services.*

Mateu, Carles. 2004. *Desarrollo de Aplicaciones Web.* Barcelona : Eureca Media, SL, 2004. ISBN: 84-9788-118-4.

Microsoft. 2011. Directorios De Servicios Web XML. *MICROSOFT.COM.* [En línea] 2011. [Citado el: 5 de 2 de 2016.] https://msdn.microsoft.com/es-es/library/7e29kfs9%28v=vs.80%29.aspx.

—. 2015. Microsoft Developer Network. [En línea] 2015. [Citado el: 30 de abril de 2016.] https://msdn.microsoft.com/es-es/library/dd409377.aspx.

MORALES MACHUCA, Carlos Andrés. 2010. *Estado del Arte: Servicios Web.* Colombia : Universidad Nacional de Colombia, Bogota, 2010.

Navarro Marset, Rafael. 2007. *REST vs Web Services.* s.l. : ELP-DSIC-UPV, 2007.

NetBeans Community. 2015. NetBeans IDE-The Smarter and Faster Way to Code. [En línea] Oracle Corporation, 2015. [Citado el: 8 de 6 de 2015.] https://netbeans.org/features/.

Perez Cuvit, Alejandro . 2011. Monografías. *http://www.monografias.com.* [En línea] 9 de 9 de 2011. [Citado el: 21 de 3 de 2016.] http://www.monografias.com/trabajos88/diagramas-clases/diagramas-clases.shtml.

Pressman, Roger S. 2012. *Ingeniería del Software, Un Enfoque Práctico, 7ma edición.* 2012.

Programmable. 2016. Programmableweb. [En línea] 2016. [Citado el: 5 de 5 de 2016.] http://programmableweb.com.

Revilla, Miguel A., Manzoor, Shahriar y Liu, Rujia. 2011. *Olympiads in Informatics Vol 2.* s.l. : Institute of Mathematics and Informatics, Vilnius, 2011.

RODRÍGUEZ SÁNCHEZ, TAMARA . 2015. *Metodología de desarrollo para la Actividad productiva de la UCI.* La Habana : UCI, 2015.

Sahni, Vinay . 2015. vinaysahni.com. [En línea] 5 de 5 de 2015. [Citado el: 6 de 5 de 2016.] http://www.vinaysahni.com/best-practices-for-a-pragmatic-restful-api.

SmartBear Software. 2016. SoapUI. *https://www.soapui.org/.* [En línea] 2016. [Citado el: 22 de 3 de 2016.] https://www.soapui.org/about-soapui/what-is-soapui.html.

Snell, James, Tidwell, Doug y Kulchenko, Pavel. 2002. *Programming Web Services with SOAP.* s.l. : O'Really, 2002. 0596000952.

Sommervilles, Ian. 2005. *Ingeniería de software 7ma edición.* s.l. : Pearson-Addison Wesley, 2005.

SparxSystems. 2007. SparxSystems. [En línea] 2007. [Citado el: 30 de abril de 2016.] http://www.sparxsystems.com.ar/resources/tutorial/uml2_deploymentdiagram.html.

Sun Microsystems, Inc. 1999. *Convenciones de código para el lenguaje de programación Java.* 1999.

The Apache Software Foundation. 2016. Apache Tomcat. *tomcat.apache.org.* [En línea] 2016. [Citado el: 9 de 2 de 2016.] http://tomcat.apache.org/.

The PostgreSQL Global Development Group. 2014. *PostgreSQL 9.3 Documentation.* Oakland : s.n., 2014.

UCID: SOFTWARE-DEFENSA. 2012. Proceso de Desarrollo y Gestión de Proyectos de Software. 2012. Vol. 1.5.

Universidad de la Ciencias Informáticas. 2016. COJ. *Caribbean Online Judge.* [En línea] 3 de 2 de 2016. [Citado el: 3 de 2 de 2016.] http://coj.uci.cu.

Van de Putte, Geert. 2004. *Using Web Services for Bussiness Integration.* s.l. : IBM, 2004. 0738425486.

Visual Paradigm. 2011. [En línea] 2011. http://www.visual_paradigm.com/product/.

Vivv, Viridiana. 2008. Ingeniería Software. [En línea] 2008. [Citado el: 29 de abril de 2016.] http://clases3gingsof.wikifoundry.com/page/Pruebas.

W3C. 2015. HTML. *W3C.* [En línea] 2015. [Citado el: 3 de 2 de 2016.] https://www.w3.org/html/.

—. 2014. Servicios Web. *W3C.* [En línea] 2014. [Citado el: 3 de 2 de 2016.] http://www.w3c.es/Divulgacion/GuiasBreves/ServiciosWeb.

—. 2016. URL. *W3C.* [En línea] 20 de 1 de 2016. [Citado el: 4 de 2 de 2016.] https://url.spec.whatwg.org/.

—. 2014. W3C - HTTP - Hypertext Transfer Protocol Overview. *W3C.* [En línea] 11 de 6 de 2014. [Citado el: 4 de 2 de 2016.] https://www.w3.org/Protocols/.

—. 2015. XML. *W3C.* [En línea] 19 de 5 de 2015. [Citado el: 3 de 2 de 2016.] https://www.w3.org/XML/.

Weerawarana, Sanjiva, Curbera, Francisco y Leymann, Frank. 2005. *Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL,WS-Reliable Messaging, and More.* s.l. : Prentice Hall PTR, 2005. 0131488740.

Yunior Mesa Reyes, Yudisney Vázquez Ortiz. 2012. *Espacio de comunicación e intercambio para la Comunidad Técnica.* La Habana : s.n., 2012.