FACULTAD 1

API REST para la gestión de los datos y servicios brindados desde el Monitor de sitios Web

Trabajo de diploma para optar por el título de
Ingeniero en Ciencias Informáticas

Autor(es): Luis Enrique Reyes Pérez
Tutor(es): M.Sc. Ariagna González Landeiro

La Habana, noviembre de 2022
"Año 63 de la Revolución"

DECLARACIÓN DE AUTORÍA

El autor del trabajo de diploma con título "***API REST para la gestión de los datos y servicios brindados desde el Monitor de sitios Web***" concede a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la investigación, con carácter exclusivo. De forma similar se declara como único autor de su contenido. Para que así conste firma(n) la presente a los 25 días del mes de noviembre del año 2022.

Luis Enrique Reyes Pérez

_____
Firma del Autor

M.Sc. Ariagna González Landeiro

_____
Firma del Tutor

AGRADECIMIENTOS

A todos los que de una forma u otra hicieron posible este logro, principalmente:

1. *A mis padres por tanto esfuerzo, paciencia, dedicación, por su guía infalible y por enseñarme el valor del trabajo, los amo.*
2. *A mi novia por sus consejos, ayuda incondicional y desinteresada, por escucharme y ser testigo de mi pasión por mi profesión a pesar de no entender ni la mitad de lo que hablo.*
3. *A mi hermana, cuñado y sobrinos por formar parte de esta aventura y su ayuda inmediata en respuesta a cualquiera de mis necesidades.*
4. *Al resto de mi familia, todos son participes de esta obra y acreedores de todos mis logros.*
5. *A mi tutora por su trabajo y tiempo empleado en esta tarea, a pesar de su apretada agenda fue una guía importante durante el proceso.*
6. *A todos mis compañeros, juntos alcanzamos este logro y no cabe duda de la influencia de cada uno en el proceso.*
7. *A todos esos profesores que todos los días lo dan todo en las aulas y contribuyen en la formación de cientos de ingenieros.*
8. *A la Universidad de Ciencias Informáticas por darme la oportunidad de cumplir mi sueño.*

RESUMEN

Desde el año 2017 se trabaja en la Universidad de las Ciencias Informáticas en el proyecto Monitor de Sitios Web, el cual tiene como objetivo principal potenciar la visibilidad en Internet de los contenidos digitales a partir de la aplicación de técnicas de posicionamiento, las cuales influyen directamente en el mejoramiento de la relevancia y popularidad de los sitios web. Para cumplir con su misión han ido resultando un conjunto de aplicaciones con un alto nivel de interrelación entre ellas. Actualmente, la comunicación se realiza de forma directa mediante la base de datos lo que propicia que no exista un adecuado sistema de conexión o intercambio de datos entre ellas. Lo antes mencionado trae consigo posibles pérdidas de clientes, atrasos en los cronogramas, aumento de los costos de producción y desmotivación de los recursos humanos. Por tanto, la presente investigación tuvo como objetivo desarrollar una API REST para la gestión de los datos y servicios brindados desde el Monitor de Sitios Web de la Universidad de Ciencias Informáticas. Para guiar el proceso de construcción de la propuesta de solución se utilizó la metodología de desarrollo de software AUP en su variante para la UCI y en la implementación se emplearon tecnologías y herramientas de software libre. La evaluación de la propuesta de solución se realizó a partir de la aplicación de técnicas y pruebas que garantizan el correcto funcionamiento de la aplicación y demostraron la satisfacción del cliente hacia la API REST desarrollada.

Palabras clave
API, API REST, servicio web, interoperabilidad.

ABSTRACT

Since 2017, the University of Informatics Sciences has been working on the Website Monitor project, whose main objective is to enhance the visibility of digital content on the Internet through the application of positioning techniques, which directly influence in improving the relevance and popularity of websites. In order to fulfill its mission, a set of applications with a high level of interrelationship between them have emerged. Currently, communication is carried out directly through the database, which means that there is no adequate connection or data exchange system between them. The aforementioned brings with it possible loss of customers, delays in schedules, increased production costs and demotivation of human resources. Therefore, the objective of this research was to develop a REST API for the management of data and services provided from the Website Monitor of the University of Informatics Sciences. To guide the construction process of the solution proposal, the AUP software development methodology was used in its variant for the UCI and free software technologies and tools were used in the implementation. The evaluation of the solution proposal was carried out from the application of techniques and tests that guarantee the correct functioning of the application and demonstrated customer satisfaction towards the REST API developed.

Keywords
API, REST API, web service, interoperability.

TABLA DE CONTENIDOS

Introducción 1

Capítulo I: Fundamentos y referentes teórico-metodológicos del desarrollo de Interfaz de Programación de Aplicaciones 6

I.1 ¿Qué es una API? 6
I.2 ¿Qué es una API web? 7
I.3 Tipos de APIs 7
I.3 Caracterización de API REST (*Representational State Transfer*) 9
I.4 Buenas prácticas para el diseño de API REST 12
I.5 Estudio de sistemas homólogos 13
I.6 Metodología de desarrollo de software 17
I.7 Lenguaje y herramienta de modelado de la solución 18
I.7.1 Lenguaje Unificado de Modelado (UML) 18
I.7.2 Visual Paradigm para UML 15.1 19
I.8 Tecnologías de desarrollo 20
I.8.1 Lenguaje de programación 20
I.8.2 Marco de trabajo 22
I.8.3 Editor de código 23
I.8.4 Javascript Object Notation (JSON) 23
I.8.5 Método de autenticación 24
I.8.6 Formato de descripción de APIs 25
I.8.7 MySQL 26
I.8.8 Servidor Web 27
I.8.9 Control de Versiones 28
Conclusiones del capítulo 29

Capítulo II: Análisis y diseño de la API REST para la gestión de los datos y servicios brindados desde el Monitor de Sitios Web 30

II.1 Propuesta de solución 30
II.2 Modelo de dominio 31
II.3 Levantamiento de requisitos 32
II.3.1 Obtención de requisitos 33
II.3.2 Especificación de requisitos de software 35
II.3.3 Descripción de historias de usuarios 37
II.3.4 Validación de requisitos de software 39
II.4 Diseño de la propuesta de solución 40
II.4.1 Diagrama de clases del diseño 41
II.4.2 Patrones de diseño 42
II.4.3 Modelado de datos 47
II.4.4 Diseño arquitectónico 49
Conclusiones del capítulo 52

Capítulo III: Implementación y evaluación de la API REST para la gestión de los datos y servicios brindados desde el Monitor de sitios Web 53
III.1 Diagrama de componentes 53
III.2 Estándares de codificación utilizados.....54
III.3 Diagrama de despliegue 56
III.4 Pruebas de software para la evaluación de la propuesta de solución.....58
III.4.1 Pruebas internas.....58
III.4.2 Pruebas de integración.....66
III.4.3 Pruebas de aceptación.....67
Conclusiones del capítulo 67
Conclusiones finales.....68
Recomendaciones.....69
Referencias bibliográficas.....70
Anexos.....75

ÍNDICE DE TABLAS

Tabla 1: Resumen de sistemas homólogos. (Fuente: Elaboración propia).....16
Tabla 2: Listado de los requisitos funcionales (Fuente: Elaboración propia) 35
Tabla 3: Listado de los requisitos no funcionales. (Fuente: Elaboración propia) 37
Tabla 4: Historia de usuario del RF5 Mostrar listado de paquetes (Fuente: Elaboración propia).....38
Tabla 5: Historia de usuario de RF 6 Mostrar Listado de subscripciones (Fuente: Elaboración propia).....38
Tabla 6: Historia de usuario de RF 7 Adicionar Suscripción (Fuente: Elaboración propia).....39
Tabla 7: Diseño de casos de prueba para el camino 3 del RF. 13 62
Tabla 8: Cantidad de no conformidades detectadas en el proceso de pruebas unitarias.....63
Tabla 9: Caso de prueba funcional HU10: Compartir sitio web 64
Tabla 10: Escenarios de caso de prueba funcional HU10 Compartir sitio web.....64
Tabla 11: Cantidad de no conformidades detectadas en el proceso de pruebas 66
Tabla 12: Cantidad de no conformidades por cada iteración de las pruebas de integración. .66

ÍNDICE DE FIGURAS

Figura 1: Diagrama de Modelo de dominio. 31
Figura 2: Diagrama de clases del diseño de las historias de usuarios 6, 7, 8, 9, 10....41
Figura 3: Patrón Controlador usado en la propuesta de solución. Método __invoke de la clase GetAll del paquete Packages....43
Figura 4: Patrón Alta cohesión usado en la propuesta de solución. Método __invoke de la clase GetCode del paquete Telus....44
Figura 5: Patrón Creador usado en la propuesta de solución. Método buildPackageApi de la clase Ge-tAll del paquete Packages. 46
Figura 6: Patrón Abstract factory usado en la propuesta de solución. La clase AppFactory del frame-work Slim. 47
Figura 7: Estructura de datos de respuesta al solicitar los grupos de variables al evaluador. 49
Figura 8: Patrón arquitectónico MVC 51
Figura 9: Arquitectura del sistema 51
Figura 10: Diseño arquitectónico de la propuesta de solución. 52
Figura 11: Diagrama de componentes de la propuesta de solución. 54
Figura 12: Ejemplo de función que cumple con todos los estándares de codificación antes listados....56
Figura 13: Diagrama de despliegue de la propuesta de solución. 57
Figura 14: Código que satisface el RF. 13 “Mostrar código de Telus” 60
Figura 15: Grafo de flujo del código que satisface el RF. 13 “Mostrar código de Telus”....60

Introducción

Un estudio anual presentado en enero de 2022 por el sitio de análisis estadísticos *We are social* («Digital Report 2022» 2022) plantea que alrededor del 62.5% de la población mundial (4.950 millones de personas) son usuarios activos de internet. Si a esto se le suma que en el 2021 la cantidad de sitios web disponibles en internet es de casi 1.880 millones (Mena Roa 2021); que cada persona encuentre lo que necesita de forma fácil y rápida es un problema difícil de solucionar. Actualmente existen varias herramientas como los directorios, meta buscadores y motores de búsquedas para facilitar el proceso de encontrar la información requerida en internet, cada una de ellas ofrece una serie de ventajas y desventajas, pero sin lugar a dudas los más consultados para esta tarea son los motores de búsquedas o buscadores debido a la gran cantidad de información que recogen, la actualización de sus bases de datos, rapidez e inmediatez en ofrecer resultados («Search engine marketing statistics 2022» 2022).

Con el crecimiento del internet en los últimos años el uso de motores de búsquedas como Google, Yahoo y Bing han sido cada vez más cruciales, su función consiste en indexar millones de sitios web y mostrar los resultados por orden de relevancia para cada consulta de búsqueda realizada (Barbar y Ismail 2019). Muchos sitios web dependen en gran medida del tráfico generado a través de un motor de búsqueda, principalmente Google, siendo este el líder en Estados Unidos (87% del mercado) y Europa (93% del mercado) para 2021 (Lewandowski, Sünkler y Yagci 2021), aunque esta no es la única fuente de tráfico hacia los sitios, también existen los tráficos directo, de pago y de referencia o social (Barbar y Ismail 2019).

A la práctica de optimizar las páginas web de una forma que mejore su ranking en los resultados orgánicos de búsqueda se le conoce como Optimización de motores de búsqueda (*Search engine optimization*, SEO) (Lewandowski, Sünkler y Yagci 2021). Es decir, aplicar modificaciones al código de los sitios web y la gestión de la visión del sitio fuera del mismo para así producir un aumento en el número de usuarios provenientes de tráfico orgánico

(entendiéndose este como el tráfico proveniente de motores de búsqueda) constituye la disciplina conocida como SEO.

En los últimos tiempos el termino SEO ha ganado un mayor significado para los diseñadores de sitios web y *Webmasters* (persona encargada de la gestión de los aspectos técnicos de un sitio web) debido al aumento del número de compañías y organizaciones que compiten por el control del mercado o la atención de los usuarios (Ebner y Granitza 2019). La complejidad de las tareas para la optimización de los sitios es elevada debido a la demanda de conocimientos técnicos necesarios para alcanzar el objetivo correcto y no caer en penalizaciones por parte de los motores de búsquedas. Esta situación condiciona que sea posible ofrecer al SEO como un servicio con el objetivo de conseguir la máxima optimización del sitio para lograr la relevancia necesaria de los términos que resultan clave para el tipo de negocio en cuestión y así gozar de una visibilidad excepcional en la página de resultados de Google (Rosado 2019).

Como parte del proceso de informatización de la sociedad y la implementación del gobierno digital Cuba ha aumentado significativamente la cantidad de sitios web gubernamentales, educacionales, de comercio electrónico, plataformas de contenidos multimedia u otro tipo de servicios disponibles a través de internet.

Con el fin, de aumentar el posicionamiento de los contenidos y servicios nacionales inicialmente y posteriormente poder exportar el servicio a sitios con dominios internacionales la Universidad de Ciencias Informáticas viene desarrollando el proyecto Monitor de Sitios Web desde el año 2017. Actualmente cuenta con cuatro aplicaciones, que van desde un directorio web, para la clasificación y categorización de los espacios digitales, un evaluador para medir el nivel de optimización de los sitios web indexador, la aplicación Telus para el análisis del tráfico que se genera hacia los medios digitales y la plataforma SeoWebMas que permite brindar el servicio online a los usuarios, dotándoles de buenas prácticas a seguir, según las evaluaciones que obtiene cada sitio web.

Estas aplicaciones tienen sus propias características y funcionalidades, no obstante, algunas de las características peculiares respecto a otras plataformas con funcionalidades similares en el mundo es que mediante una sola aplicación (SeoWebMas) el usuario puede recibir el resultado del diagnóstico SEO y el análisis de tráfico web.

Existe una alta interoperabilidad entre las aplicaciones, principalmente de SeoWebMas con el resto de las aplicaciones. No obstante, pudiera pasar que como parte de la evolución del proyecto siguiera creciendo la necesidad de intercambio de datos entre otras aplicaciones del proyecto que puedan ir surgiendo, o entre las ya existentes. Además, hay una demanda real de un grupo de clientes que desean mostrar en sus sitios web algunos de los datos gestionados por las aplicaciones del Monitor.

Actualmente la forma más común de interoperar las aplicaciones es mediante consultas directa a las bases de datos. Por ejemplo, en el caso de SeoWebMas, esto propicia que la aplicación opere con la estructura interna de cada base de datos y su correcto funcionamiento se ponga en riesgo ante cambios que se realicen en las misma, lo que evidencia limitaciones en la capacidad de comunicación entre los componentes del software, además del incremento en la complejidad del desarrollo del lado de la aplicación. Esto sumado a la falta de preparación de los recursos humanos, donde la mayoría están en proceso de formación, y deben a su vez duplicar roles de desarrollo hace que complejice el proceso de desarrollo de nuevas funcionalidades y consigo la evolución de los productos y servicios del proyecto. Esto se traduce en atrasos en los cronogramas de desarrollo y limitada capacidad de respuesta ante solicitudes de los clientes afectándose las oportunidades de negocio del proyecto, así como el tiempo y el presupuesto disponible para el mismo.

Se define como **Problema de investigación**: ¿Cómo mejorar la capacidad de comunicación entre los componentes de software de las aplicaciones que conforman el proyecto Monitor de Sitios Web para lograr mayor interoperabilidad entre ellas y con terceras aplicaciones?

Todo lo dicho anteriormente conduce a plantearnos como **objeto de estudio**: El Proceso de gestión de servicios web, enfocándonos en el **campo de acción** las aplicaciones del proyecto Monitor de Sitios web cubanos de la Universidad de Ciencia Informáticas.

Para dar solución a la problemática planteada en el presente documento se plantea como **objetivo general** de este trabajo de diploma: Implementar una API REST para la gestión de los datos y servicios brindados desde las aplicaciones del Monitor del Sitio Web.

Del objetivo general anterior podemos desglosar los siguientes **objetivos específicos**:

- Establecer los fundamentos teórico-metodológicos de la gestión de servicios web.
- Realizar un estudio de las tendencias en el proceso de desarrollo de APIs
- Implementar la API REST para la gestión de los datos y servicios brindados desde las aplicaciones del Monitor del Sitio Web.
- Validar la solución lográndose una mejor interoperabilidad entre las aplicaciones del Monitor.

Se definen como **preguntas científicas:**

1. ¿Cuáles son los presupuestos teóricos del desarrollo de interfaces de programación de aplicaciones (API)?
2. ¿Qué aspectos se deben tener en cuenta para diseñar una API REST?
3. ¿Cuáles son las herramientas y tecnologías más adecuadas para la implementación de una API REST que gestione los datos y servicios brindados desde las aplicaciones del Monitor del Sitio Web?
4. ¿Qué pruebas de software aplicar para la evaluación de la API REST para la gestión de los datos y servicios brindados desde las aplicaciones del Monitor del Sitio Web?

En el transcurso de la investigación se utilizaron los siguientes métodos de investigación:

Métodos teóricos:

- **Analítico-Sintético:** permitió descomponer el problema en partes, pudiendo realizar un mejor análisis, se consultó distintas fuentes bibliográficas de información y se

extrajo los principales elementos asociados al tema de desarrollo de Interfaces de programación de aplicaciones (API).

- **Histórico-Lógico:** con el objetivo de constatar teóricamente cómo ha evolucionado el desarrollo de API's, los estilos existentes, así como las herramientas y tecnologías más utilizadas en la actualidad.
- **Inductivo-Deductivo:** permitió integrar los métodos y funcionalidades, así como los patrones de diseño para resolver problemas particulares de la solución en desarrollo, facilitando el análisis de los elementos generales a elementos más específicos.

Métodos empíricos:

- **Entrevistas:** para obtener las principales características de las aplicaciones del proyecto, enfatizando en sus estructuras de datos, y otros aspectos importantes que contribuyeran a conocer más cada una de estas.
- **Observación:** permitió conocer la arquitectura interna de las aplicaciones del Monitor de Sitios Web cubanos, sus componentes, mecanismos de comunicación y estructuras de datos.

Capítulo I: Fundamentos y referentes teórico-metodológicos del desarrollo de Interfaz de Programación de Aplicaciones

En el presente capítulo se hace un estudio de los elementos involucrados en el proceso de integración de datos en aplicaciones informáticas mediante el uso de APIs, los diferentes tipos de APIs que existen, mejores prácticas de diseño y necesidades de documentación de las mismas. Por otra parte, se caracterizan la metodología, tecnologías y herramientas a utilizar durante el ciclo de desarrollo de la solución propuesta.

I.1 ¿Qué es una API?

API, acrónimo de *Application Programming Interface* que significa Interfaz de Programación de Aplicaciones, es un concepto que se puede definir de diferentes formas con pequeñas variaciones cada una. Investigadores como (ARNAUD 2019) la definen como una interface expuesta por algún software, una abstracción de la implementación que está por detrás de alguna funcionalidad lo que permite convertir el software en bloques reusables que pueden ser utilizados por otros para hacer sistemas modulares y su objetivo es permitirle a las personas alcanzar su objetivo de la forma más simple posible.

Otros autores la definen como un programa intermediario que permite que dos aplicaciones se comuniquen entre sí, un mecanismo que toma una solicitud de un usuario, le dice al sistema que acción realizar y devuelve una respuesta al usuario. Una API define funcionalidades que son independientes de su respectiva implementación lo que les permite cambiar su implementación sin afectar a los consumidores de la API (Cleveland et al. 2020).

Ambas definiciones coinciden en que son un conjunto de funciones, reglas o procedimientos utilizados a modo de abstracción que permite a los consumidores acceder a las funcionalidades de determinado sistema sin conocer los detalles de implementación, una vía de intercambio de información.

I.2 ¿Qué es una API web?

Una API web como su nombre lo indica es una API, pero con ligeras características que las diferencian de las APIs convencionales como pudieran ser las descritas por (GEEWAX 2021):

- Utiliza una red de datos como medio de comunicación.
- Puede ser utilizada por diferentes personas al mismo tiempo.
- Los desarrolladores de la API web tienen más control que los que la usa.
- No pueden ser usadas de forma local, descargando la librería o usándola de forma independiente.
- Los usuarios están obligados a aceptar los cambios en las funcionalidades, sean requeridos o no.

I.3 Tipos de APIs

Existen diferentes formas de comunicación entre aplicaciones sobre una red, pero todas persiguen el mismo objetivo de intercambiar datos y ofrecer funcionalidades, lo que cambia es la forma en que lo hacen y la tecnología que las soporta. Algunas de estas formas son estilos arquitectónicos, otros son protocolos o lenguajes de consultas, de forma general pueden ser clasificadas como estilos de APIs, como se hace en (ARNAUD 2019). A continuación, se listan algunos de los estilos más utilizados y una corta caracterización de los mismos.

Remote Procedure Call (RPC)

Es un protocolo que provee un paradigma de comunicación de alto nivel usado en el sistema operativo. RPC asume la existencia de un protocolo de transporte de bajo nivel como podría ser *Transmission Control Protocol/Internet Protocol* (TCP/IP) o *User Datagram Protocol* (UDP) que se encargue de enviar los datos entre los programas que se comunican. RPC se sustenta sobre la lógica de los sistemas cliente-servidor diseñados específicamente para soportar aplicaciones de red («IBM Docs - RPC» 2022).

gRPC

Marco de trabajo de código abierto para llamadas a procedimientos remotos (RPC) de propósito general a través de una red. Desarrollado por Google en 2015 como una variante de la arquitectura RPC, construido sobre HTTP\2.0(*Hyper Text Transfer Protocol*) y sigue un modelo de comunicación cliente-respuesta (Lee y Liu 2022). Permite la llamada a procedimientos remotos como si estuviera llamando a procedimientos locales en sistemas conectados sin necesidad de programar detalles de la interacción remota. Soporta diferentes formatos de transferencia de datos como JSON(*Javascript Object Notation*), XML(*Extensible Markup Language*), Protobuf (Predeterminado, debido a su alto rendimiento) y Thrift (Sharma 2021).

***Simple Object Access Protocol* (SOAP)**

Protocolo de mensajes que permite que programas escritos en diferentes lenguajes de programación se comuniquen con facilidad usando XML como mecanismo de comunicación y sobre el protocolo HTTP. Es una plataforma independiente que ofrece características como seguridad, privacidad, calidad de la transmisión de datos y confiabilidad (Hussein 2021).

GraphQL

Lenguaje de consultas declarativas y de manipulación de datos, un entorno de ejecución del lado de servidor para APIs. Permite a los clientes consultar los datos exactos que desean, ni menos, ni más. GraphQL maneja 3 conceptos importantes, consultas, subscripciones y mutaciones, todos son definidos en el esquema de GraphQL; usando una sola ruta de acceso (endpoint) es capaz de responder usando JSON basado en la solicitud del cliente, que puede ser una consulta, mutación o suscripción (Sharma 2021). Fue desarrollada por Facebook en 2012 y liberada en 2015, no está atada a ninguna base de datos ni motor de almacenamiento de datos, en su lugar está respaldada por el código y los datos existente («GraphQL | A query language for your API» 2022).

***Representational State Transfer* (REST)**

Estilo arquitectónico que propone un conjunto de restricciones y reglas para la creación de APIs, lo que permite que las aplicaciones se comuniquen con otras. Utiliza los métodos HTTP para realizar sus operaciones de comunicación y JSON/XML como lenguaje de formateo de texto.

Haber realizado un estudio de diferentes estilos de APIs permitió al autor de la siguiente investigación evaluar las características y particularidades de cada uno y sus casos de uso. Para el desarrollo de la propuesta de solución y dado que todas las aplicaciones del proyecto Monitor de Sitios Web son basadas en la web, entonces se consideró que el estilo arquitectónico REST debe ser el que más se ajusta. Por tal razón en el siguiente epígrafe se caracteriza con mayor nivel de profundidad el estilo seleccionado.

.

I.3 Caracterización de API REST (*Representational State Transfer*)

Como anteriormente se definió, REST es un estilo arquitectónico para aplicaciones web. Fue ideado por Roy Fielding en su tesis doctoral y utiliza los métodos HTTP para realizar sus operaciones de comunicación (solicitud y respuesta) para luego retornar una respuesta en formato JSON/XML al cliente (Hussein 2021). Como estilo arquitectónico está diseñado para ayudar a crear y organizar sistemas distribuidos, siendo un error verlo como un estándar, una guía o algo que implicaría que hay un conjunto de reglas estrictas que seguir para obtener una arquitectura RESTful (Doglio 2018).

Según diferentes autores como (Doglio 2018), (Sharma 2021), (Hussein 2021) y (Cleveland et al. 2020) existen algunos criterios que toda aplicación debería cumplir para ser considerada RESTful:

- Cliente-Servidor: Una arquitectura cliente-servidor permite la separación de responsabilidades. El cliente es responsable de solicitar y mostrar los datos, mientras que el servidor maneja el almacenamiento de datos y la lógica de la aplicación.
- Sin estados (*Stateless*): La comunicación entre el cliente y el servidor es *stateless*, lo que significa que cada solicitud del cliente contiene toda la información necesaria para que el servidor la procese.

- Cache: La cache permite que los clientes sean capaces de almacenar las respuestas del servidor y reutilizarlas en solicitudes similares, disminuyendo el tráfico de red y los tiempos de espera.
- Interfaz Uniforme: Es una de las características más destacada y ganadora si se compara con otras alternativas. Mantener una interface uniforme entre componentes simplifica el trabajo del cliente cuando viene a interactuar con el sistema. Otro punto de vista provechoso es la separación de la implementación del cliente de la del sistema, por lo que mantener una interface uniforme simplifica el trabajo de los desarrolladores del lado del cliente dándoles un grupo de reglas claras que seguir.
- Sistema de capas: El diseño de REST está pensado para trabajar apropiadamente en aplicaciones de red, por lo que una aplicación de este tipo debería ser capaz de manejar grandes cantidades de datos. Separar los componentes del sistema en capas, permitirle a cada una usar solamente la inmediata inferior y devolver su salida a la superior simplifica la complejidad del sistema y permite verificar el correcto funcionamiento de cada componente.
- Código bajo demanda: Es una característica opcional, lo que significa que no es necesaria para que una aplicación sea considerada RESTful. Se refiere a la capacidad del cliente de descargar y ejecutar código del servidor.

Una de las características distintivas de REST como estilo arquitectónico es la utilización de métodos HTTP para especificar el tipo de acción que será realizada sobre algún recurso, a continuación, se listan los más importantes según (Doglio 2018):

- GET: Accede a un recurso en modo solo lectura.
- POST: Se usa normalmente para enviar nueva información al servidor, es una acción de creación.
- PUT: Usado para actualizar algún recurso del servidor.
- DELETE: Usado para eliminar recursos del servidor.
- HEAD: Usado para preguntar sobre la existencia de algún recurso sin devolver su representación.
- OPTIONS: Usado para solicitar el listado de verbos disponibles sobre algún recurso.

- PATCH: Realiza una actualización parcial de recursos. Mezclar un recurso dado con datos nuevos recibidos.

Otra de las características de HTTP de las que REST se beneficia es la utilización de códigos de estado (número que resume la respuesta asociada a él). Son particularmente útiles para que los clientes tengan una rápida interpretación de la respuesta, pero no debería ser una sustituta para la misma. Los códigos de estado pueden agruparse en 5 clases («Códigos de estado de respuesta HTTP - HTTP | MDN» 2022):

- Respuestas informativas (100–199),
- Respuestas satisfactorias (200–299),
- Redirecciones (300–399),
- Errores del lado del cliente (400–499),
- Errores del lado del servidor (500–599).

La caracterización antes hecha sobre REST deja una clara idea de lo que debería tener un sistema que pretenda utilizar este estilo arquitectónico y ser considerado RESTful, pero la decisión de adoptar este estilo debe ser respaldada por hechos o razones, por lo que a continuación se listan algunas de las mencionadas en (Doglio 2018):

- Rendimiento: El estilo de comunicación propuesto por REST está pensado para que sea simple y eficiente, permitiendo un aumento en el rendimiento de los sistemas que lo adoptan.
- Escalabilidad de la interacción entre componentes: El simple hecho de disponer de un sistema de capas distribuidos permite esta característica de forma natural, lo que facilita el desarrollo y mantenimiento del sistema.
- Interfaz simple: Una interfaz simple permite una simple interacción entre sistemas, haciendo intuitiva la utilización de la API.
- Fácil modificación de componentes: La naturaleza distribuida del sistema y la separación de responsabilidades les permite a los desarrolladores modificar los componentes de forma independiente y con un mínimo riesgo.

- Portabilidad: Significa que es independiente de cualquier tecnología o lenguaje de programación, pudiendo ser implementado o consumido en cualquiera de ellos.
- Confiable: La característica sin estado (*Stateless*) de REST permite la fácil recuperación del sistema tras un fallo.

I.4 Buenas prácticas para el diseño de API REST

El diseño de APIs es una práctica compleja que no debe ser tomada a la ligera. Durante el desarrollo de una API REST existen varios factores que se deben tener en cuenta, no solo un código limpio, buenas prácticas de desarrollo u otras consideraciones internas, por lo que varios autores proponen una serie de requerimientos durante el diseño de una API REST para obtener un buen producto. A continuación, se detallan algunas de las más importantes según (Sharma 2021), (GEEWAX 2021), (ARNAUD 2019) y (Doglio 2018):

De forma general se puede decir que una API REST debe satisfacer ciertos criterios como los siguientes para ser considerada una buena API:

- Amigable para desarrolladores (*Developer-friendly*): Los desarrolladores que utilizan la API deben sentirse cómodos con ella.
- Extensible: El sistema debe soportar la adición de nuevas características sin que los clientes se vean afectados.
- Documentación al día: Una buena documentación es imprescindible para que una API sea elegida por nuevos desarrolladores.
- Correcto manejo de errores: Los errores son una parte intrínseca del software, por lo que estar preparados nunca está de más.
- Proveer múltiples SDK/Librerías: Mientras más trabajo se simplifique a los consumidores mejor aceptada será la API.
- Seguridad: Es un aspecto fundamental en cualquier sistema.

Para cumplir con algunos de los anteriores criterios se propone el siguiente grupo de buenas prácticas, encaminadas principalmente en mejorar la interacción del cliente con la API, su seguridad y rendimiento:

1. Usar sustantivos en vez de verbos cuando se nombra un recurso en la ruta de acceso(*endpoint*). Es redundante usar verbos puesto que los métodos HTTP usados en REST los usan.
2. Usar el plural cuando nombramos una colección de recursos en la ruta de acceso (*endpoint*)
3. Usar Hipermedia (enlaces a otros recursos) hace el trabajo de cliente fácil, proporcionando enlaces a otros recursos en las respuestas, eliminando la necesidad de que el cliente cree la ruta y automatizando la actualización en caso de que ocurra algún cambio en la misma.
4. El versionado de la API es una característica clave que permite que los clientes sigan utilizando la versión que más le convenga sin que esto afecte la adición de nuevas funcionalidades.
5. La seguridad en las APIs es crucial, por lo que requiere una atención especial. Algunas recomendaciones serían la utilización de HTTPS (HTTP seguro) y un sistema de autenticación como JWT (JSON web token) o OAuth 2.0.
6. La documentación debe estar disponible en todo momento y actualizada con las últimas modificaciones y el correcto versionado.
7. Usar correctamente los verbos HTTP y los códigos de estados propuestos por el estilo.
8. Usar correctamente el sistema de cache, HTTP provee los mecanismos de cacheo de información, el desarrollador solo debe proveer las cabeceras correctas en la respuesta de la API para que el cliente pueda sacar provecho de esto.
9. Establecer un límite de peticiones mediante el uso de las cabeceras de la respuesta para prevenir el uso excesivo de la API y de ser necesario utilizar el método HTTP 429 *Too Many Requests*.

I.5 Estudio de sistemas homólogos

En la actualidad se puede encontrar una API en cualquier lugar, desde APIs proporcionadas por dispositivos para facilitar el uso de funcionalidades como el manejo de la cámara, el GPS o acceso al almacenamiento interno hasta APIs dispersas por todo el internet donde las instituciones y empresas ofrecen todo tipo de servicios. En este estudio de homólogos se

estarán revisando algunas de las principales APIs disponibles en internet para su consulta y uso en la obtención o procesamiento de datos.

YouTube

La API de datos de esta aplicación permite a los usuarios integrar su programa con YouTube y realizar muchas de las operaciones disponibles en el sitio web. Brinda la capacidad de buscar videos, recuperar fuentes estándar y ver contenidos relacionados. Un programa también puede autenticarse como usuario para cargar videos, modificar listas de reproducción de usuarios y más. Esta API RESTful proporciona respuestas en formato JSON («YouTube» 2022).

LinkedIn

LinkedIn es el centro de redes sociales de negocios más grande del mundo. La API de LinkedIn es una plataforma RESTful que proporciona una representación simple y consistente de personas, empresas, trabajos y las interacciones y relaciones entre ellos. Utiliza XML y JSON como lenguaje de formato de la información y OAuth 2.0 para autorizar a los usuarios y comenzar a realizar llamadas a la API REST utilizando cualquier lenguaje de programación. El acceso a la API está restringido a desarrolladores autorizados («LinkedIn» 2022).

AmazonS3

Amazon Web Services (AWS) proporciona una plataforma de infraestructura confiable y de bajo costo que impulsa a cientos de miles de empresas. El Servicio de Almacenamiento Simple (*Simple Storage Service*) proporciona una interfaz de servicios web simple que se utiliza para almacenar objetos utilizando la infraestructura de almacenamiento en línea de Amazon. La API está basada en REST. Las respuestas están formateadas en JSON («Amazon S3» 2022).

API REST de Google *PageSpeed Insights*

La API de Google *PageSpeed Insights* permite a los desarrolladores medir mediante programación el rendimiento de una página web y obtener sugerencias sobre cómo mejorar el rendimiento, la accesibilidad y el SEO de esa página. Esta API devuelve datos del Informe de experiencia del usuario de Chrome y datos de laboratorio de Lighthouse. Los desarrolladores

necesitarán una clave API. La API utiliza OAuth 2 como mecanismo de autenticación y JSON como formato de los datos de respuesta, siendo un API tipo REST.

API GraphQL de PayPal Braintree

Empresa estadounidense de alcance mundial que opera un sistema de pagos en línea que soporta transferencias de dinero entre usuarios y sirve como una alternativa electrónica a los métodos de pago tradicionales como cheques y giros postales. La API esta construida con el lenguaje de consultas declarativas GraphQL, utiliza API Key como mecanismo de autenticación y retorna una respuesta en formato JSON («Braintree GraphQL API» 2022).

Netflix API

Netflix es un servicio de *streaming* por suscripción que les permite a sus miembros ver series y películas sin publicidades en un dispositivo con conexión a internet. En Netflix se utiliza una API basada en el protocolo gRPC de Google para la comunicación entre sus componentes de software internos. La API utiliza un mecanismo de autenticación basado en tokens y el lenguaje Protobuf como medio de comunicación (Netflix Technology 2021).

API GraphQL de Font Awesome

Font Awesome es una biblioteca de iconos de código abierto y kit de herramientas que se utiliza para 100 millones de sitios web. Está basado en CSS y Less. La API GraphQL de Font Awesome permite a los desarrolladores consultar y recuperar iconos y datos con GraphQL. La API utiliza API Key como mecanismo de autenticación y devuelve respuestas en formato JSON.

API REST asociada repositorio de Nova

API REST que ofrezca información asociada al uso del repositorio de la distribución cubana GNU/Linux Nova. La API permite mostrar la información de los paquetes que contiene el repositorio, además contribuye a prestar un mejor servicio en cuanto a la información asociada a los repositorios. Para la comunicación se utiliza JSON como lenguaje de intercambio de datos y utiliza un sistema de autenticación basado en token.

Existe un número no despreciable de importantes empresas que utilizan las APIs para brindar sus servicios, las APIs no están orientadas a un área en particular, lo que les permite estar presente en casi todas las aplicaciones que se usan actualmente. Es importante recalcar que el número de APIs tenidas en cuenta al realizar el actual estudio de sistemas homólogos es extenso, el autor considera que los sistemas presentados en el presente documento son los más significativos. A continuación, se presenta una tabla resumen donde se exponen las principales características identificadas en el estudio de sistemas homólogos realizado.

Tabla 1: Resumen de sistemas homólogos. (Fuente: Elaboración propia)

Empresa	**API**	**Modo de autenticación**	**Estilo Arquitectónico**	**Formato de respuesta**
Google	YouTube API	API Key	REST	JSON
LinkedIn	LinkedIn API	OAuth 2	REST	JSON, KML, XML
Amazon	Amazon S3	API Key	REST	JSON, XML
Google	Google Page Speed Insights	OAuth 2	REST	JSON
PayPal	Braintree's GraphQL API	API Key	GraphQL	JSON
Netflix	Netflix API	Token	gRPC	Protobuf
Fonticons	Font Awesome GraphQL API	API Key	GraphQL	GraphQL, JSON
Centro de Software Libre.	API REST asociada repositorio de Nova	Token	REST	JSON

Una vez analizados los datos anteriores se llegó a la conclusión de que existe una amplia variedad de tipos de APIs que son utilizadas de forma activa en la actualidad. Es necesario resaltar que las APIs, independientemente del estilo que se utilice para su desarrollo están centradas en las características y necesidades del negocio al que le corresponde por lo que el resultado de este estudio de sistemas homólogos estaría centrado sobre la viabilidad de utilizar una API para solventar los problemas de intercambio de datos de las aplicaciones del Monitor

de Sitios Web, la selección del estilo a utilizar, así como sus características. La investigación realizada validó que el uso del estilo arquitectónico REST tiene alta vigencia en aplicaciones de internet. Además, que el formato de respuesta JSON es viable, sencillo de utilizar y tiene una práctica de uso probado en aplicaciones de uso nacional e internacional.

I.6 Metodología de desarrollo de software

En todo proceso de desarrollo de software es imprescindible contar con un nivel de planificación y control que permita monitorear el avance del proyecto a la par que se verifique que se está avanzando en la dirección correcta. Según (Rodríguez Sánchez 2015) una metodología de desarrollo de software es un marco de trabajo usado para estructurar, planificar y controlar el proceso de desarrollo de un sistema informático. Es un conjunto de procedimientos, técnicas, artefactos y herramientas que guían a los desarrolladores en la realización de software.

En la actualidad se cuenta con varios enfoques para el proceso de desarrollo de software, uno de ellos es el enfoque tradicional o pesado, el cual propone una amplia documentación y planeación de todo el proyecto, es utilizado en proyectos de gran envergadura y equipos de desarrollo grande; otro enfoque seria el ágil, como su nombre indica propone mecanismos más ágiles y enfocados a la rápida producción del software, útil en equipos más pequeños, con buen conocimiento de las tecnologías y una lógica del negocio no muy compleja.

Dentro del enfoque ágil existen muchas metodologías a escoger como son: Programación extrema (XP), Proceso Unificado Ágil (AUP), Proceso Unificado Ágil versión UCI (AUP-UCI) entre otros. Todas ellas ofrecen una serie de ventajas y desventajas, luego de realizarse un análisis de las mismas se decidió escoger AUP-UCI para el desarrollo de la solución propuesta en este documento. Para el proceso de obtención de requisitos la metodología seleccionada propone varios escenarios, en este trabajo se utiliza el escenario número 4 puesto que, al existir una lógica de negocios definida y modelada previamente para el desarrollo del sistema ya existente, el proyecto no posee gran envergadura, el equipo de desarrollo es pequeño y se

contará con la participación activa del cliente en el proceso, se dan las condiciones idóneas para la utilización del mismo.

La metodología Variación de AUP para la UCI está forma por tres fases, (Inicio, Ejecución y Cierre) para el ciclo de vida de los proyectos de la universidad, las cuales contienen las características de las cuatro fases (Inicio, Elaboración, Construcción y Transición) propuestas en AUP. Las características de las fases de la metodología de la universidad son (Rodríguez Sánchez 2015):

- **Inicio:** durante el inicio del proyecto se llevan a cabo las actividades relacionadas con la planeación del proyecto. En esta fase se realiza un estudio inicial de la organización cliente que permite obtener información fundamental acerca del alcance del proyecto, realizar estimaciones de tiempo, esfuerzo y costo y decidir si se ejecuta o no el proyecto.
- **Ejecución:** en esta fase se ejecutan las actividades requeridas para desarrollar el software, incluyendo el ajuste de los planes del proyecto considerando los requisitos y la arquitectura. Durante el desarrollo se modela el negocio, se obtienen los requisitos, se elaboran la arquitectura y el diseño, se implementa y se libera el producto.
- **Cierre:** en esta fase se analizan tanto los resultados del proyecto como su ejecución y se realizan las actividades formales de cierre del proyecto.

I.7 Lenguaje y herramienta de modelado de la solución

Para modelar la propuesta de solución a la problemática planteada en el presente trabajo se utilizó Lenguaje Unificado de Modelado (UML) como lenguaje de modelado y la herramienta Visual Paradigm para UML en la creación de los diferentes tipos de diagramas presentes en este documento. A continuación, se dan elementos complementarios sobre la utilización de esta herramientas y lenguaje.

I.7.1 Lenguaje Unificado de Modelado (UML)

El Lenguaje Unificado de Modelado es un estándar en la industria del desarrollo de software, notación (esquemática en su mayor parte) con que se construyen sistemas por medio de conceptos orientados a objetos (Larman 1999). Permite a los creadores de sistemas generar

sistemas que capturen sus ideas en una forma convencional y fácil de comprender para comunicarlas a otras personas. UML es la clave para organizar el proceso de diseño de forma tal que los analistas, clientes, desarrolladores y otras personas involucradas en el desarrollo del sistema lo comprendan y convengan con él (Schmuller 1999).

Es un lenguaje para especificar, visualizar, construir y documentar los artefactos de los sistemas de software, así como para el modelado del negocio. Posee vocabularios y reglas que se centran en la representación conceptual y física de un sistema, para crear y leer modelos bien formados (Larman 1999). En este trabajo investigativo se utiliza UML para visualizar, documentar y construir la propuesta de solución, lo que supone una ventaja al momento de compartirla con otros desarrolladores o cuando sea necesario aplicar mantenimiento al sistema.

I.7.2 Visual Paradigm para UML 15.1

Visual Paradigm es una herramienta Computer Aided Software Engineering (CASE, por sus siglas en inglés) que emplea UML como lenguaje de modelado. Las herramientas de Ingeniería de Software Asistida por Computadora se pueden definir como un conjunto de programas y ayudas que dan asistencia a los analistas, desarrolladores e ingenieros de software, durante el ciclo de vida de un software, proporcionándole un aumento en su productividad y logrando un mayor ahorro de tiempo (Ambler 2017).

Visual Paradigm soporta el ciclo de vida de desarrollo de software completo, desde el análisis y diseño hasta la construcción, pruebas e incluso despliegue. Permite representar gráficamente varios diagramas y facilita la generación de código. Es una herramienta multiplataforma, fácil de instalar y utilizar. A continuación se listan algunas de sus principales ventajas según el sitio oficial («Visual Paradigm» 2022):

- Soporte nativo de UML
- Diagramas de procesos de negocio
- Diagramas entidad relación y transformación en tablas de bases de datos
- Soporte a técnicas de ingeniería inversa

- Capacidad de convertir un modelo en código

I.8 Tecnologías de desarrollo

Para llevar a cabo el desarrollo de la propuesta de solución se utilizaron un conjunto de tecnologías ampliamente utilizadas en el mundo del desarrollo de software. En los siguientes subepígrafes se hace una caracterización de cada una de ellas.

I.8.1 Lenguaje de programación

Un lenguaje de programación es un sistema de comunicación que posee una determinada estructura, contenido y uso. La programación es el procedimiento de escritura del código fuente de un software. De esta manera, puede decirse que la programación le indica al programa informático qué acción tiene que llevar a cabo y cuál es el modo de concretarla. El lenguaje de programación tiene la capacidad de especificar, de forma precisa, cuáles son los datos que debe trabajar un equipo informático, de qué modo deben ser conservados o transferidos dichos datos y qué instrucciones debe poner en marcha la computadora ante ciertas circunstancias (Pérez Porto y Merino 2012). Para la solución propuesta se decidió utilizar como lenguaje de programación php en su versión 7.4.3, a continuación, se muestran algunas de sus características.

Lenguaje de programación PHP 7.4.3

PHP (*Hypertext Preprocessor*) es lo que se conoce como un lenguaje de comandos (*scripting languag*e) del lado de servidor, lo que significa que está escrito para automatizar ciertas tareas y ejecutarse en el servidor en vez de en la computadora del cliente. Algunas estimaciones indican que PHP soporta un tercio de la web. A pesar de la complejidad de las páginas web construidas con él, es fácil de aprender para programadores nuevos a la par de ofrecer varias características avanzadas para los más experimentados.

Fue creado por Rasmus Lerdorf en 1994 como herramienta de automatización para su página personal, a día de hoy es considerado como uno de los lenguajes de programación más

utilizado, de acuerdo con *W3Techs Web Technology Surveys* es usado en el 82.6 % de los sitios cuyo lenguaje de programación pudo ser identificado (Murphy 2018).

PHP fue originalmente creado para crear páginas web, a lo largo de los años ha madurado en un poderoso lenguaje cuya popularidad es resultado de su gran cantidad de beneficios (Murphy 2018):

- Tecnología de código abierto y soporte de la comunidad: Una de sus mejores características es ser de código abierto, disponible para desarrolladores y usuarios que quieran aportar en su desarrollo o simplemente usarlo.
- Multiplataforma: PHP trabaja en múltiples plataformas, esto incluye tanto las de código abierto como las propietarias. También es compatible con una amplia variedad de servidores web con diferentes programas servidores y es fácil encontrar un proveedor de alojamiento de servidores en internet.
- Comandos del lado del servidor (Server-side Scripting): Otra de las mejores características es la capacidad de ejecutar comandos del lado del servidor, lo que elimina la necesidad de que el cliente tenga habilitado que los comandos puedan ejecutarse en su computadora.
- Simplicidad: PHP fue originalmente escrito en C y sus sintaxis resulta muy parecidas. Se le hace muy fácil a los desarrolladores aprender lo básico y empezar a programar debido a que PHP tiene una curva de aprendizaje muy sencilla.
- Versatilidad: PHP no solamente está destinado a trabajar con HTML, PHP puede procesar imágenes, archivos pdf, videos flash y una gran variedad de archivos de texto como XML o XHTML. Por otra parte, PHP también soporta gran cantidad de protocolos de programación como HTTP, Lightweight Directory Access Protocol (LDAP), File Transfer Protocol (FTP), Internet Message Access Protocol (IMAP)
- Rendimiento: Los desarrolladores de PHP han trabajado arduamente a lo largo de los años para mejorar su rendimiento y algunas grandes empresas como Facebook han estado trabajando en una versión que dicen ser capaz de aumentar la velocidad de ejecución de código hasta 6 veces.

- Soporte de Bases de datos: Otra gran característica es la gran cantidad de bases de datos soportadas por PHP. Cualquier aplicación PHP puede acceder a una base de datos de diferentes formas como puede ser MySQLi o usando PHP Data Objects (PDO).
- Librerías: Desde que PHP es popular cuenta con una gran cantidad de desarrolladores creando librerías para facilitar el proceso de desarrollo. Debido a la gran cantidad de librerías desarrolladas existen herramientas automatizadas para su gestión y control en proyectos.

I.8.2 Marco de trabajo

Según (Murphy 2018) un marco de trabajo web (Web Framework) es un conjunto de métodos estandarizados para construir y organizar aplicaciones web. Un recurso para ayudar a los desarrolladores a construir aplicaciones web de forma más fácil y eficiente.

Slim Framework 4.4

Slim es un micro-framework que sirve para construir rápidamente aplicaciones web y APIs simples pero poderosas. Slim es usado a día de hoy para conectar múltiples aplicaciones que no pueden compartir los mismos recursos. Está diseñado para optimizar el código PHP y desarrollar aplicaciones web usando una arquitectura MVC (*Model View Controller*) (Sunardi y Suharjito 2019). Dentro de las características más destacadas de Slim se encuentran («Slim Framework» 2022):

- Enrutador HTTP: Slim provee un enrutador rápido y poderoso que mapea las funciones (*callbacks*) a métodos HTTP de solicitudes específicos y rutas. El enrutador también soporta parámetros y patrones.
- Middleware: permite crear aplicaciones con middlewares concéntricos para modificar los objetos de solicitud y respuesta HTTP en torno a su aplicación Slim.
- Compatibilidad con PSR-7: Slim admite cualquier implementación de mensajes HTTP PSR-7, por lo que puede inspeccionar y manipular el método, el estado, el URI, los encabezados, las cookies y el cuerpo del mensaje HTTP.
- Inyección de dependencia: Slim admite la inyección de dependencia para que tenga un control completo de sus herramientas externas.

I.8.3 Editor de código

Los editores de código permiten editar código fuente en diversos lenguajes de programación y ofrecen múltiples herramientas para facilitar el trabajo y aumentar la productividad. Generalmente son programas ligeros, que ofrecen lo necesario para poder ser productivos y tener una experiencia de desarrollo adecuada, pero sin complicaciones. Sin embargo, los editores actuales se pueden extender tanto como se quiera, por medio de complementos que los pueden hacer llegar a ser tan avanzados como los IDE («Editores de código» 2022).

Visual Studio Code

Es un editor de código multiplataforma, de código abierto y gratis desarrollado por Microsoft. Tiene soporte nativo para diferentes lenguajes de programación como JavaScript, Typescript, CSS, SASS y tecnologías como JSON, HTML, entre otros. Dentro de sus características destaca el soporte *IntelliSense*, con resaltado de sintaxis, autocompletado y detección de errores. Está equipado con utilidades para la detección de errores en el funcionamiento del código (*debug*) e integración completa con Git, permitiendo controlar las versiones del proyecto de forma transparente y fácil. Posee capacidad de incorporar nuevas funcionalidades instalando extensiones desarrolladas por la comunidad o por equipos de desarrollo profesionales («Visual Studio Code» 2022).

I.8.4 Javascript Object Notation (JSON)

JSON es considerado como un formato estándar de transferencia de datos que ha ganado cierta popularidad en los últimos tiempos, en parte debido a las ventajas que provee (Doglio 2018):

- Ligero: Hay muy pocos datos en un archivo JSON que no esté relacionado con la información que está siendo transferida.
- Entendible por humanos: Es un lenguaje tan simple que puede ser leído y escrito por humanos sin mucho trabajo, este es un punto importante considerando que el punto importante de la interfaz de la API es el factor humano (*Developer Experience*, DX).

- Tipos de datos: Soporta diferentes tipos de datos, permitiendo añadir significado a los datos que están siendo transferidos.
- Parseo: Fácil de parsear y generar por las computadoras, lo que aumenta la rapidez de generación y lectura, reflejándose en tiempos de respuesta menores.

I.8.5 Método de autenticación

Dentro de los mecanismos más importantes en cuanto a seguridad de una API REST se encuentra el método de autenticación empleado para garantizar que un cliente es realmente quien dice ser antes del intercambio de datos que pudieran comprometer determinados aspectos relacionados con la seguridad y privacidad de la información. En la actualidad existen una gran variedad de formas de autenticar solicitudes a APIs, a continuación se listan algunas de las más usadas según («Most Popular API Authentication Methods | 3Pillar Global» [sin fecha]):

- Autenticación básica por HTTP: La forma más simple de manejar la autenticación de una API, envía los datos de usuario y contraseña en cada solicitud.
- Autenticación por API Key: Este método crea una llave (*key*) única para los desarrolladores, la cual debe ser enviada en cada solicitud. No existe una longitud estándar para la llave, pero como mínimo debe ser una cadena de texto de al menos 30 caracteres alfanuméricos.
- Autenticación por OAuth: OAuth es el protocolo estándar de la industria para la autorización. OAuth se enfoca en la simplicidad del desarrollador del cliente al tiempo que proporciona flujos de autorización específicos para aplicaciones web, de escritorio y móviles.
- Autenticación por JWT: JSON Web Token (JWT) es un estándar abierto (RFC 7519) que define una forma compacta y autónoma de transmitir información de forma segura entre las partes como un objeto JSON. Esta información es verificable y confiable debido a que está firmada digitalmente.

Para el desarrollo de la propuesta de solución se optó por el mecanismo de autenticación por JWT debido a su simplicidad, eficiencia y reducción del tamaño de los datos transmitidos en

cada solicitud en comparación con otros mecanismos existentes. Además, la posibilidad de cifrar los datos utilizando pares de claves públicas y privadas es una mejora considerable en la seguridad de la información, que al final estará circulando por internet a merced de los atacantes. Por último, otra de las características más atractivas en cuanto a facilidad de desarrollo está la capacidad de convertir los datos a objetos de forma sencilla, debido a que al final del día JWT usa JSON como formato y la gran mayoría de lenguajes de programación ofrecen facilidades de mapeo de JSON a objetos de forma nativa (auth0.com 2022).

I.8.6 Formato de descripción de APIs

A la hora de describir una interfaz de programación y especialmente sus datos es más simple y eficiente utilizar una herramienta estructurada como un formato de descripción de APIs. Algunas de las ventajas que ofrecen este tipo de herramientas es la facilidad de compartir el diseño del sistema con cualquiera que se involucre en el proyecto y puede ser fácilmente comprendido por las personas que conozcan el formato o cualquier herramienta de generación automática de documentaciones.

OpenAPI Specification (OAS)

La especificación OpenAPI (OAS) define una interfaz estándar independiente del idioma para las API RESTful que permite que tanto los humanos como las computadoras descubran y comprendan las capacidades del servicio sin acceso al código fuente, la documentación o a través de la inspección del tráfico de red. Cuando se define correctamente, un consumidor puede comprender e interactuar con el servicio remoto con una cantidad mínima de lógica de implementación. Las herramientas de generación de documentación pueden usar una definición de OpenAPI para mostrar la API, herramientas de generación de código para generar servidores y clientes en varios lenguajes de programación, herramientas de prueba y muchos otros casos de uso.

Dentro del estándar existen una serie de definiciones como los documentos, donde se definen o describen las APIs, las plantillad de ruta, secciones de una URL encerradas dentro de llaves ({}) donde se definen parámetros que son reemplazables por valores reales, tipos de medio,

sección donde se describe el tipo de dato que será transmitido, pudiendo ser JSON (application/json), texto plano (text/plain; charset=utf-8) o cualquier otro de los definidos por la especificación RFC6838 (Freed, Klensin y Hansen 2013) y por último los códigos de estados HTTP que como ya se ha dicho son usados para indicar la naturaleza de la respuesta de determinada operación sobre algún recurso.

La especificación utiliza un versionado semántico (por ejemplo 3.0.1) donde 3.0 representa las versiones mayor y menor respectivamente y el 1 los parches de seguridad o correcciones de errores. Los documentos de descripción pueden ser escritos utilizando JSON o YAML indistintamente («OpenAPI Specification» 2022).

OAS fue previamente conocida como *Swagger Specification* y es perfectamente compatible con las herramientas Swagger, un conjunto de utilidades de código abierto que facilitan el proceso de desarrollo, las tres principales herramientas que serán utilizadas son:

- *Swagger Editor*: Usado para diseñar, describir y documentar las APIs, dedicado específicamente para las APIs basadas en la especificación OpenAPI («Swagger Editor» 2022).
- *Swagger Codegen*: simplifica el proceso de compilación al generar stubs de servidor y SDK de cliente para cualquier API, definida con la especificación OpenAPI, permitiendo que el equipo pueda concentrarse mejor en la implementación y adopción de la API («Swagger Codegen» 2022).
- *Swagger UI*: permite que cualquier persona, ya sea del equipo de desarrollo o consumidor, visualice e interactúe con los recursos de la API sin tener implementada ninguna lógica de implementación. Se genera automáticamente a partir de la especificación OpenAPI («Swagger UI» 2022).

I.8.7 MySQL

MySQL es el sistema de gestión de bases de datos relacional más extendido en la actualidad al estar basada en código abierto. Desarrollado originalmente por MySQL AB, fue adquirida por Sun MicroSystems en 2008 y está su vez comprada por Oracle Corporation en 2010, la

cual ya era dueña de un motor propio InnoDB para MySQL. Es un sistema de gestión de bases de datos que cuenta con una doble licencia. Por una parte, es de código abierto, pero por otra, cuenta con una versión comercial gestionada por la compañía Oracle (Robledano 2019).

Dentro de las principales capacidades de MySQL se encuentran:

- Arquitectura Cliente y Servidor
- Compatibilidad con SQL
- Compatibilidad con vistas personalizadas
- Procedimientos almacenados
- Desencadenantes
- Transacciones

I.8.8 Servidor Web

Los servidores web (también conocidos como servidores HTTP) son un tipo de servidores utilizados para la distribución de contenido web en redes internas o en Internet. Como parte de una red de ordenadores, un servidor web transfiere documentos a los llamados clientes. Los servidores web sirven para almacenar contenidos de Internet y facilitar su disponibilidad de forma constante y segura. Un servidor web permite el cifrado de la comunicación entre el servidor web y el cliente, autenticación HTTP para áreas específicas de una aplicación web y almacenamiento en caché de documentos dinámicos para la respuesta eficiente de solicitudes y evitar una sobrecarga del servidor («Servidor web - IBM Docs» 2022).

Apache

Apache es un servidor web HTTP de código abierto. Está desarrollado y mantenido por una comunidad de usuarios en torno a la Apache Software Foundation. Actualmente y desde el 1996 es el servidor web más usado en todo el mundo debido a su seguridad y estabilidad. Apache tiene una estructura basada en módulos, que permite activar y desactivar funcionalidades adicionales. También permite ajustar los parámetros de PHP de tu hosting de forma personalizada mediante el fichero de configuración (.htaccess) («¿Qué es Apache y para qué sirve?» 2019).

Las principales ventajas de usar este el servicio web son las siguientes:

- De código abierto y gratuito, con una gran comunidad de usuarios.
- Parches de seguridad regulares y actualizados con frecuencia.
- Estructura basada en módulos.
- Multiplataforma. Está disponible en servidores Windows y Linux.
- Personalización mediante .htaccess independiente en cada hosting.

I.8.9 Control de Versiones

Para gestionar las versiones del código, los cambios y errores se decidió utilizar Git como sistema de control de versiones y Gitlab como repositorio, a continuación, se brindan algunos aspectos sobre ellos.

Git

Sistema de control de versiones utilizado para rastrear cambios en archivos de computadora. El propósito principal de Git es administrar los cambios realizados en uno o más proyectos durante un período de tiempo determinado. Ayuda a coordinar el trabajo entre los miembros de un equipo de proyecto y realiza un seguimiento del progreso a lo largo del tiempo. Git también ayuda tanto a los profesionales de la programación como a los usuarios no técnicos al monitorear sus archivos de proyecto. Git puede manejar proyectos de cualquier tamaño. Permite que varios usuarios trabajen juntos sin afectar el trabajo de los demás («What is Git» 2022).

Gitlab

Repositorio de Git basado en la web que proporciona repositorios abiertos y privados gratuitos, capacidades de seguimiento de problemas y wikis. Es una plataforma DevOps completa que permite a los profesionales realizar todas las tareas de un proyecto, desde la planificación del proyecto y la gestión del código fuente hasta la supervisión y la seguridad. Además, permite que los equipos colaboren y creen un mejor software.

El principal beneficio de usar Gitlab es que permite que todos los miembros del equipo colaboren en cada fase del proyecto. Ofrece seguimiento desde la planificación hasta la creación para ayudar a los desarrolladores a automatizar todo el ciclo de vida de DevOps y lograr los mejores resultados posibles. Cada vez más desarrolladores han comenzado a usar Gitlab debido a su amplia variedad de características y disponibilidad de bloques de código («What is GitLab and How to Use It? [2022 Edition] | Simplilearn» 2022).

Conclusiones del capítulo

Haber realizado un estudio de los conceptos relacionados con el objeto de estudio permitió establecer los elementos esenciales a tener en cuenta durante el desarrollo de la solución. El estudio realizado sobre los diferentes tipos de APIs existentes permitió evaluar las características de cada uno y los beneficios de usarlo. La caracterización realizada sobre la API REST ayudó a identificar las características de este estilo arquitectónico, como funciona y que criterios propone para ser considerada RESTful. Al realizar el estudio de sistemas homólogos se detectó que muchas empresas modernas de gran valor utilizan el estilo arquitectónico REST y formato de transferencia de datos JSON para brindar sus servicios o datos a los consumidores. La profundización en los referentes teóricos del desarrollo de los diferentes tipos de APIs, así como el estudio de diversas aplicaciones que utilizan este mecanismo de comunicación permitió validar el uso de estas para mejorar la interoperabilidad entre las aplicaciones. El realizar un profundo análisis del objeto de estudio y el profundizar en las características de las aplicaciones que conforman el Monitor de Sitios Web permitió identificar los aspectos que se deben tener en cuenta para diseñar un API REST, así como la metodología de desarrollo, las tecnologías y herramientas a utilizar.

Capítulo II: Análisis y diseño de la API REST para la gestión de los datos y servicios brindados desde el Monitor de Sitios Web

En el presente capítulo se describen la propuesta de solución, las tareas de ingeniería realizadas y los productos de trabajo obtenidos durante la ejecución de las disciplinas de requisitos, análisis y diseño. Los principales productos obtenidos en esta fase de desarrollo son diagrama de clases del dominio, los requerimientos funcionales y no funcionales que debe cumplir la API REST de SeoWebMas, así como las Historias de usuarios correspondientes. Por último, se describe la arquitectura del sistema a desarrollar, los patrones de diseño utilizados y los diagramas de clases del sistema y de despliegue.

II.1 Propuesta de solución

Dadas las necesidades planteadas en la situación problemática del presente trabajo de diploma, se determina desarrollar una API REST que tribute a la solución de los problemas de interoperabilidad entre las aplicaciones del proyecto Monitor de Sitios Web cubanos. Los datos a integrar están basados en los servicios ofrecidos por la interface web SeoWebMas y serán transmitidos en formato JSON para ser consumidos como sean necesarios.

La solución propuesta estará basada en la arquitectura cliente servidor, el cliente podrá enviar solicitudes con instrucciones sobre los datos a obtener y el servidor las procesará y dará una respuesta de forma transparente y sin necesidad de que el cliente tenga conocimiento de cómo se hizo. Se tendrán en cuenta tanto los códigos de estado como los verbos HTTP en las solicitudes a la hora de realizar las peticiones sobre algún recurso. El sistema contará con una capa de seguridad que actuará como autenticador de usuarios y control de acceso a los recursos protegidos, para su implementación se usará un *middleware* que implemente el estándar JWT (JSON Web Tokens). Por último, para mejorar la experiencia de los consumidores del servicio se documentará todo el sistema y los recursos compartidos utilizando la especificación OpenAPI.

II.2 Modelo de dominio

Para identificar los principales conceptos del mundo real involucrados en la situación problemática, la relación entre ellos, personas involucradas o eventos existentes en los procesos desarrollados, así como alcanzar un nivel de comprensión alto sobre los mismos y establecer un lenguaje común en el contexto del sistema se utilizó un diagrama de modelo de dominio, una herramienta de la disciplina de ingeniería de software que según (Larman 1999) se define como una representación de las clases conceptuales del mundo real, no de componentes de software. No se trata de un conjunto de diagramas que describen clases de software, u objetos de software con responsabilidades, sino que modela clases conceptuales significativas en un determinado problema.

A continuación, se muestra el diagrama de modelo de dominio de la situación problemática existente:

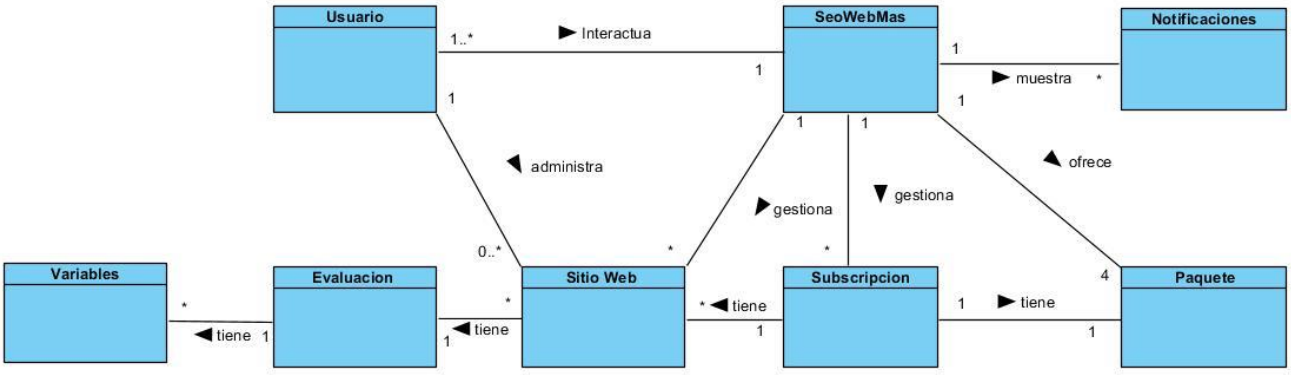

Figura 1: Diagrama de Modelo de dominio.

(Fuente: Elaboración Propia)

Descripción del modelo de dominio

- **Usuario:** Persona que interactúa con SeoWebMas para conocer el estado SEO de los sitios web que administra.
- **SeoWebMas:** Interface web que brinda los servicios del proyecto Monitor de Sitios Web cubanos, permitiéndole a los usuarios diagnosticar el estado de los sitios que administra.

- **Evaluación:** Resultado del proceso realizado por la aplicación Evaluador de sitios web sobre alguno de los sitios gestionados por el proyecto.
- **Sitio web:** Representa un sitio web que es administrado por un usuario y diagnosticado por el proyecto.
- **Suscripción:** Acuerdo económico entre el proyecto y un usuario en el que se especifican los datos legales, sitios a ser procesados por el proyecto y el pago por los servicios del proyecto.
- **Notificaciones:** Información de importancia que el proyecto desea que el usuario conozca mediante SeoWebMas.
- **Paquete:** Oferta de los servicios disponibles del proyecto para ser adquirida por un usuario mediante una suscripción.

En el diagrama de Modelo de dominio se establecen relaciones entre los conceptos involucrados, dando a entender ideas tales como que un usuario con suscripción en SeoWebMas que administra sitios web puede acceder a los servicios del monitor mediante subscripciones, en cada una de ellas se ofrece un paquete con diferentes ofertas de los servicios a recibir si opta por la misma y posibilitándole acceder a un número finito de variables que serán medidas y evaluadas en concordancia con los sitios web que posee dicho usuario. Por otra parte, los usuarios de SeoWebMas reciben notificaciones con distintas informaciones, dichas notificaciones pueden ser públicas o privadas.

Una vez identificados y definidos los elementos que intervienen en la situación problemática, las relaciones que existen entre ellos e implicaciones que traen dichas relaciones, se procede a la obtención de requisitos de software bajo el escenario 4 de la metodología AUP versión UCI empleada en la presente investigación.

II.3 Levantamiento de requisitos

Los requisitos de un sistema son la descripción de lo que es sistema debe hacer, los servicios que proporciona y las restricciones de sus operaciones. Dichos requerimientos reflejan las necesidades de los clientes del sistema. El proceso de obtención, análisis, documentación y

validación de esos servicios y restricciones se llama ingeniería de requisitos. Los requisitos se clasifican en funcionales (servicios que el sistema debe proporcionar) y no funcionales (restricciones de los servicios o funciones ofrecidos por el sistema) (Sommerville 2011). A continuación, se describe cómo se obtuvieron los requisitos de la propuesta de solución, su especificación, descripción y validación.

II.3.1 Obtención de requisitos

Las técnicas de identificación de requisitos de software según (Sommerville 2011) son procesos para la obtención de información sobre el sistema requerido y sistemas existentes, permite separar los requerimientos del sistema y los del usuario. Algunas fuentes de información son documentaciones, *stakeholders* y especificaciones de sistemas similares. (Pressman 2010) indica que estas técnicas permiten identificar las necesidades de negocio de los clientes y los usuarios. Son mecanismos que se utilizan para recolectar la información necesaria en la obtención de los requisitos de una aplicación, permiten investigar aspectos generales para posteriormente ser especificados con un mayor detalle, requieren ser adecuadamente orientadas para cubrir la información que se requiere capturar.

Observación

Permite obtener información de primera mano sobre cómo se efectúan las actividades mediante la observación de las formas en que se llevan a cabo los procesos, además posibilita verificar que realmente se sigan todos los pasos especificados para una determinada tarea (Guerra 2017). Durante el proceso de obtención de requisitos de la propuesta de solución, la técnica de observación se empleó para conocer cómo se llevan a cabo los principales procesos dentro del proyecto, principalmente la suscripción, gestión de paquetes, vías de interacción con usuarios y el proceso de evaluación de un sitio web. En el anexo 1 se presenta la observación realizada a los procesos del proyecto Monitor de Sitios Web cubanos.

Entrevista

Fuente de información de gran utilidad para obtener información cualitativa como opiniones, o descripciones subjetivas de actividades. Es una técnica muy utilizada, y requiere una mayor

preparación y experiencia por parte del analista. La entrevista se puede definir como un “intento sistemático de recoger información de otra persona” a través de una comunicación interpersonal que se lleva a cabo por medio de una conversación estructurada. Debe quedar claro que no basta con hacer preguntas para obtener toda la información necesaria. Es muy importante la forma en que se plantea la conversación y la relación que se establece en la entrevista (Guerra 2017).

El anexo 2 de este documento es la entrevista realizada a los especialistas del proyecto Monitor de Sitios Web cubanos, la misma permitió obtener información sobre los principales datos de SeoWebMas a integrar en los servicios ofrecidos por la propuesta de solución, algunos datos nuevos a integrar y la forma en que SeoWebMas integra los servicios del proyecto.

Estudio de documentación

El estudio de varios tipos de documentación, como manuales y reportes, pueden proporcionar al analista información valiosa con respecto a las organizaciones y a sus operaciones. La documentación difícilmente refleja la forma en que realmente se desarrollan las actividades, o donde se encuentra el poder de la toma de decisiones. Sin embargo, puede ser de gran importancia para introducir al analista al dominio de operación y el vocabulario que utiliza (Guerra 2017). El anexo 3 presenta la guía de estudio de la documentación del proceso de suscripción, la cual describe cómo se lleva a cabo el proceso de suscripción, los datos que se recopilan y como se procesan para concluir con un usuario satisfecho que consume los servicios del proyecto y una compensación económica a cambio.

Las técnicas llevadas a cabo para la recopilación de información relacionada con los procesos del proyecto Monitor de Sitios Web cubanos, así como los datos y servicios brindados a través de SeoWebMas junto a las opiniones de los especialistas del proyecto esclarecieron aspectos relacionados con la ingeniería de requisitos, específicamente la fase de obtención de requisitos permitiendo al autor de este trabajo de diploma captar las principales necesidades de software y humanas en las que se debe centrar, las mismas serán presentadas en el epígrafe siguiente.

II.3.2 Especificación de requisitos de software

Una especificación de requerimientos de software (ERS) es un documento que se crea cuando debe especificarse una descripción detallada de todos los aspectos del software que se va a elaborar, antes de que el proyecto comience (Pressman 2010). Los requisitos de software de la propuesta de solución se presentan a continuación.

Requisitos funcionales

Como se dijo anteriormente, los requisitos funcionales son los servicios que debe proporcionar el sistema, son la descripción de lo que este debe hacer. La tabla número 2 muestra los requisitos funcionales de la propuesta de solución, una descripción de cada uno y la complejidad de cada uno según el producto de trabajo Evaluación de requisitos, propuesto en el expediente de proyecto 5.0 para la actividad productiva de la universidad.

Tabla 2: Listado de los requisitos funcionales (Fuente: Elaboración propia)

Número	Requisito Funcional	Descripción	Complejidad
RF. 1	Mostrar sitios web	Permite mostrar un listado con los sitios web que el usuario tiene con una suscripción aprobada, los sitios que le han sido compartido y los sitios en los que cuenta con permisos especiales.	Media
RF. 2	Mostrar evaluación de sitio web.	Permite mostrar la evaluación de un sitio web desglosada por variables, cada una evaluada, medido su impacto, complejidad, identificado el grupo al que pertenece y recomendaciones para mejorar.	Alta
RF. 3	Mostrar notificaciones públicas	Permite mostrar las notificaciones públicas del sistema y que aún no estén vencidas.	Baja
RF. 4	Mostrar notificaciones privadas	Permite mostrar las notificaciones de un usuario en particular y que aún no estén vencidas.	Baja
RF. 5	Mostrar listado de paquetes	Permite mostrar el listado de paquetes disponibles junto a su descripción y características.	Baja

RF. 6	Mostrar listado de suscripciones	Permite mostrar el listado de subscripciones del usuario, identificando el tipo de suscripción, nombre de la entidad o persona subscrita, precio, periodo de vigencia y cantidad de sitios subscritos.	Media
RF. 7	Adicionar nueva suscripción	Permite al usuario realizar una suscripción de sus sitios a alguno de los paquetes existentes. Incluye cálculo del precio y la notificación del estado de la suscripción.	Alta
RF. 8	Editar suscripción	Permite al usuario editar los datos de una suscripción.	Alta
RF. 9	Renovar suscripción	Permite al usuario renovar una suscripción que haya terminado su tiempo de vigencia.	Alta
RF. 10	Obtener suscripción	Permite que el usuario obtenga los datos de una suscripción especificada.	Media
RF. 11	Compartir sitio web	Permite que el usuario comparta un sitio web de una de sus subscripciones con otro usuario especificado mediante un correo electrónico, incluye la notificación del mismo.	Alta
RF. 12	Autenticar usuarios	Permite autenticar a los usuarios que consuman del servicio.	Alta
RF. 13	Mostrar código de Telus	Permite mostrar el código de Telus de un sitio web.	Alta
RF. 14	Notificar Usuarios mediante emails	Permite notificar a los usuarios a los que se les comparte algún servicio por parte de otro usuario del sistema.	Media
RF. 15	Notificar al personal mediante emails	Permite notificar al personal del Monitor de Sitios Web en caso de haber una suscripción nueva o se manifieste alguna anomalía en el proceso.	Media

Requisitos no funcionales

Definir requisitos no funcionales implica establecer las restricciones sobre los servicios ofrecidos por el sistema, en función de las necesidades del usuario. La tabla número 3 especifica los requisitos no funcionales de la propuesta de solución según las características o atributos de calidad propuestos en el producto de trabajo Especificación de requisitos de software del expediente de proyecto 5.0 para la actividad productiva de la universidad.

Tabla 3: Listado de los requisitos no funcionales. (Fuente: Elaboración propia)

Número	Tipo	Requisito No Funcional
RNF. 1	Disponibilidad	El sistema debe estar disponible todo el tiempo para los usuarios autorizados.
RNF. 2		El período entre fallos recuperables, como por ejemplo fallos en el servidor no debe exceder las 24 horas.
RNF. 3	Eficiencia	Los tiempos de respuesta deben estar por debajo de los 3 segundos.
RNF. 4	Usabilidad	El sistema está creado para ser utilizado por personas con conocimientos de informática, las respuestas serán en formato JSON con las claves en inglés.
RNF. 5	Diseño e implementación	Como lenguaje de programación para la interfaz web se deberá utilizar PHP en su versión 7.4.3.
RNF. 6		Para el desarrollo de la aplicación web se deberá utilizar Slim en su versión 4.

II.3.3 Descripción de historias de usuarios

En el escenario 4 para la obtención de requisitos de la metodología AUP-UCI se emplean las historias de usuarios como mecanismos de descripción de los requisitos funcionales. Las historias de usuarios son escritas por el cliente, en su propio lenguaje, como descripciones cortas de lo que el sistema debe realizar. Cada historia de usuario debe ser lo suficientemente comprensible y delimitada para que los programadores puedan estimar el tiempo de desarrollo e de implementarlas sin dificultad (Joskowicz 2008). A continuación, se muestran algunas de

las historias de usuario de diferentes requisitos de la propuesta de solución con diferentes niveles de complejidad, las demás historias de usuario están en el anexo 4.

Tabla 4: Historia de usuario del RF5 Mostrar listado de paquetes (Fuente: Elaboración propia)

Historia de Usuario	
Número: 5	**Nombre de la historia:** Mostrar listado de paquetes
Programador: Luis Enrique Reyes Pérez	**Iteración Asignada:** 1
Prioridad en negocio: Alta	**Tiempo estimado:** 1 semana
Riesgo en desarrollo: No aplica	**Tiempo real:** 1 semana
Descripción: Permite mostrar el listado de paquetes disponibles junto a su descripción y características. Se consulta la base de datos y se extraen los datos necesarios.	
Observaciones: • Para acceder a la URL debe utilizar el método GET.	

Tabla 5: Historia de usuario de RF 6 Mostrar Listado de subscripciones (Fuente: Elaboración propia)

Historia de Usuario	
Número: 6	**Nombre de la historia:** Mostrar listado de suscripciones
Programador: Luis Enrique Reyes Pérez	**Iteración Asignada:** 1
Prioridad en negocio: Alta	**Tiempo estimado:** 2 semana
Riesgo en desarrollo: No aplica	**Tiempo real:** 2 semana
Descripción: Permite mostrar el listado de subscripciones del usuario, identificando el tipo de suscripción, nombre de la entidad o persona subscrita, precio, periodo de vigencia y cantidad de sitios subscritos. Se consulta la base de datos y se piden los datos de las subscripciones del usuario autenticado junto a la cantidad de sitios de cada una.	
Observaciones: • El usuario debe estar autenticado • Para acceder a la URL debe utilizar el método GET.	

Tabla 6: Historia de usuario de RF 7 Adicionar Suscripción (Fuente: Elaboración propia)

Historia de Usuario	
Número: 7	**Nombre de la historia:** Adicionar suscripción
Programador: Luis Enrique Reyes Pérez	**Iteración Asignada:** 1
Prioridad en negocio: Alta	**Tiempo estimado:** 2 semana
Riesgo en desarrollo: No aplica	**Tiempo real:** 2 semana
Descripción: Permite al usuario realizar una suscripción de sus sitios a alguno de los paquetes existentes. Incluye cálculo del precio y la notificación del estado de la suscripción. Se recibe un listado con los sitios a añadir a la suscripción, el paquete contratado, así como una serie de datos necesarios para procesar la suscripción. Se debe verificar que los sitios recibidos estén en el directorio de no estarlo se le notificara al usuario mediante correo y a los encargados del centro para su procesamiento y posterior adición al mismo. Si todos los datos están correctos se hace un cálculo del precio, se crea la suscripción y notifica al usuario del éxito del proceso mediante correo electrónico.	
Observaciones: • El usuario debe estar autenticado • Para acceder a la URL debe utilizar el método POST.	

II.3.4 Validación de requisitos de software

La calidad de los productos del trabajo que se generan como consecuencia de la ingeniera de los requerimientos se evalúa durante el paso de validación. La validación de los requerimientos analiza la especificación a fin de garantizar que todos ellos han sido enunciados sin ambigüedades; que se detectaron y corrigieron las inconsistencias, las omisiones y los errores, y que los productos del trabajo se presentan conforme a los estándares establecidos para el producto (Pressman 2010). La validación de requisitos es el proceso de verificar que los requerimientos describen el sistema que el cliente realmente quiere, se superpone con el análisis y se encarga de encontrar problemas con los requisitos (Sommerville 2011). Las técnicas de validación de requisitos utilizadas en la propuesta de solución son descritas a continuación.

Diseño de casos de pruebas

Los diseños de casos de pruebas permiten crear un conjunto de entradas y salidas esperadas que sean efectivos para descubrir defectos en los programas y muestren que el sistema satisface sus requerimientos. Para su diseño, se selecciona una característica del sistema o componente que se está probando, un conjunto de entradas que ejecutan dicha característica, se documentan las salidas esperadas o rasgos de salida y donde sea posible se diseña una prueba automatizada que demuestre que las salidas reales y las esperadas son las mismas (Sommerville 2011). Los casos de pruebas elaborados para la validación de los requisitos funcionales de la propuesta de solución se muestran en el epígrafe 3.4.

Revisión Técnica Formal

Una revisión técnica formal (RTF) es una actividad del control de calidad del software realizada por ingenieros de software. Los objetivos de una RTF son: descubrir los errores en funcionamiento, lógica o implementación de cualquier representación del software; verificar el cumplimiento de sus requisitos; garantizar su representación de acuerdo a los estándares predefinidos; obtener uniformidad en su desarrollo y hacer proyectos más manejables (Pressman 2010). La RTF realizada a los diferentes productos de trabajo obtenidos en esta disciplina fue desarrollada por uno de los especialistas del Monitor de Sitios Web.

II.4 Diseño de la propuesta de solución

En esta fase de desarrollo los requisitos pueden ser refinado y estructurados para conseguir una mejor comprensión de los mismos con una descripción que sea fácil de mantener y soporte la estructura del sistema. En esta disciplina se modela el sistema y su forma (incluida su arquitectura) para que soporte todos los requisitos, incluyendo los requisitos no funcionales (Rodríguez Sánchez 2015). El diseño de software agrupa el conjunto de principios, conceptos y prácticas que llevan al desarrollo de un sistema o producto de calidad, permite modelar el sistema que se va a construir proporcionando detalles de la arquitectura del software, estructuras de datos, interfaces y componentes que se necesitan para implementar el sistema (Pressman 2010). En los siguientes subepígrafes se hace una definición de los elementos del análisis y diseño considerados para la implementación de la propuesta de solución.

II.4.1 Diagrama de clases del diseño

El diagrama de clases del diseño describe gráficamente las especificaciones de las clases de software y de las interfaces en una aplicación, es un conjunto de definiciones de entidades software más que de conceptos del mundo real. Las clases del diagrama pueden tener relaciones entre ellas tales como relaciones de dependencia, asociativas y herencia (Larman 1999). A continuación, se muestra el diagrama de clases del diseño correspondiente a las historias de usuarios 6, 7, 8, 9, 10.

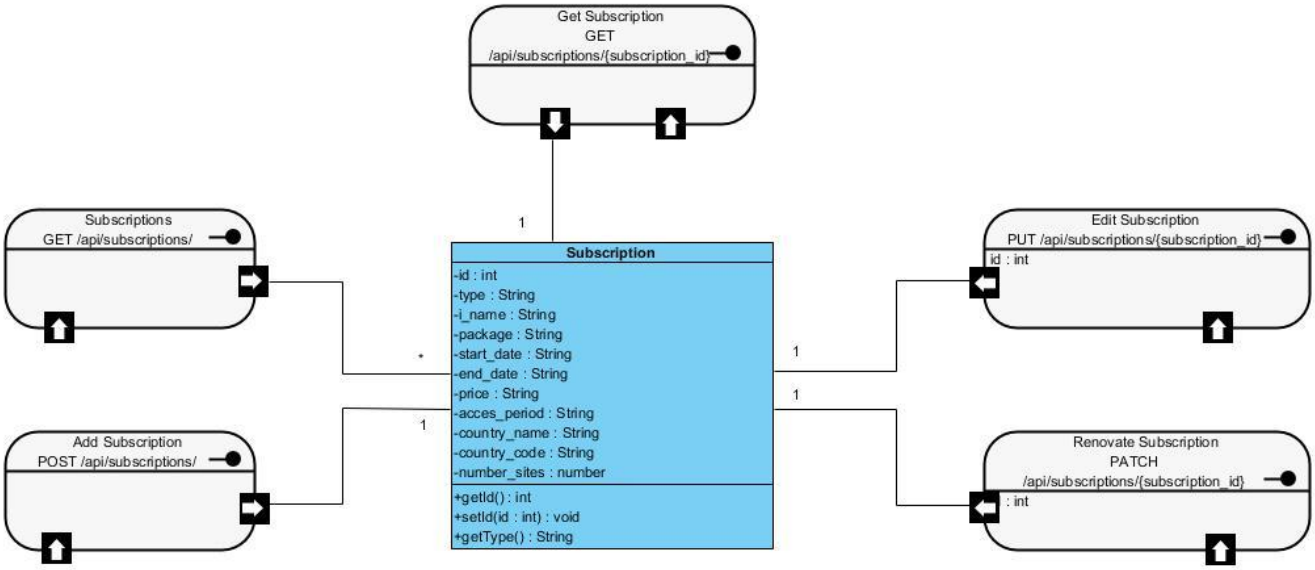

Figura 2: Diagrama de clases del diseño de las historias de usuarios 6, 7, 8, 9, 10
(Fuente: Elaboración propia)

Descripción del diagrama de clases del diseño

- *Subscriptions*: Encargada de listar las subscripciones del usuario.
- *Add Subscription*: Encargada de crear una suscripción nueva del usuario autenticado.
- *Edit Subscription*: Encargada de editar una suscripción existente del usuario autenticado especificada por el id.
- *Renovate Subscription*: Encargada de renovar una suscripción vencida del usuario autenticado especificada por el id.
- *Get Subscription*: Encargada de obtener una suscripción del usuario autenticado especificada por el id y devolverla.

El diagrama de clases del diseño antes mostrado se centra en la clase entidad Suscripción, representada con todos sus atributos y métodos. Los requisitos funcionales 6, 7, 8, 9 y 10 estarían satisfaciéndose mediante los *endpoints* a implementar representados en el diagrama, cada uno de ellos sería una clase con la lógica necesaria para satisfacer un requisito, en el diagrama se especifica la ruta a consultar, los parámetros necesarios y el método HTTP que sería necesario utilizar.

II.4.2 Patrones de diseño

Un patrón de diseño describe una estructura de diseño que resuelve un problema particular del diseño dentro de un contexto específico y entre fuerzas que afectan la manera en la que se aplica y en la que se utiliza dicho patrón. El objetivo de cada patrón es proporcionar una descripción que permita a un diseñador determinar si el patrón es aplicable al trabajo en cuestión, si puede volverse a usar y si sirve como guía para desarrollar un patrón distinto en función o estructura (Pressman 2010). El patrón es una descripción de un problema y su solución que recibe un nombre y que puede emplearse en otros contextos; en teoría, indica la manera de utilizarlo en circunstancias diversas (Larman 1999).

En resumen, un patrón de diseño es una estructura de diseño que describe un problema y su solución práctica, es un estándar que se utiliza al diseñar sistemas que resuelven determinados problemas comunes y que ayuda a un mejor entendimiento del código por parte de otros programadores.

Patrones GRASP

Los patrones GRASP describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones. GRASP es un acrónimo que significa *General Responsibility Assignment Software Patterns* (patrones generales de software para asignar responsabilidades). El nombre se eligió para indicar la importancia de captar (*grasping*) estos principios, si se quiere diseñar eficazmente el software orientado a objetos (Larman 1999).

Controlador:

Un Controlador es un objeto de interfaz no destinada al usuario que se encarga de manejar un evento del sistema (evento de alto nivel generado por un actor externo). Define además el método de su operación (Larman 1999). En la propuesta de solución se utilizó este patrón mediante las clases que responden a las solicitudes del usuario, dichas clases implementan el método *__invoke* donde se especifica el comportamiento que manejara el evento, en la figura se observa una implementación del método *__invoke* que satisface la historia de usuario 5, retornando el listado de paquetes disponibles.

```php
public function __invoke(Request $request, Response $response): Response
{
    /** @var Medoo $db */
    $db = $this->container->get('em');

    $sql = 'select id, name, description, feature_pricing_tables, show_in_home,
            show_btn_home_quota, show_btn_home_subscription, show_in_pricing_tables
            from package;';

    $result = $db->query($sql)->fetchAll();
    $data = array();
    if (count($result)) {
        foreach ($result as $p) {
            $data[] = $this->buildPackageApi($p);
        }
    }
    return $this->responseHttpOk($response, $data);
}
```

Figura 3: Patrón Controlador usado en la propuesta de solución. Método __invoke de la clase GetAll del paquete Packages.

(Fuente: Elaboración propia)

Alta cohesión

En la perspectiva del diseño orientado a objetos, la cohesión es una medida de cuán relacionadas y enfocadas están las responsabilidades de una clase. Una alta cohesión caracteriza a las clases con responsabilidades estrechamente relacionadas que no realicen un trabajo enorme, debe existir una colaboración entre las clases para compartir el esfuerzo y no caiga todo el peso sobre una única clase. Usar este patrón simplifica el mantenimiento y

favorece el bajo acoplamiento (Larman 1999). En la propuesta de solución se utilizó este patrón en la clase Controladora GetCode del paquete Telus que satisface la historia de usuario 13. El método __invoke de la clase GetCode maneja el evento de una solicitud y la clase MatomoService se encarga de la comunicación con el servicio remoto de Telus.

```php
public function __invoke(Request $request, Response $response, array $args): Response
{
    /** @var Medoo $db*/
    $db = $this->container->get('em');
    // Seleccionar los atributos del sitio pasado por los argumentos
    // El sitio debe ser del usuario autenticado.
    $sql = 'Select w.id, w.host, w.homepage, w.host_telus
            from website w join website_permission wp on w.id = wp.website_id
            where w.id = \'' . $args['website'] . '\'
            and wp.owner = \'' . $this->getUserNameJwt($request) . '\'';
    $result = $db->query($sql)->fetch();
    // Retornar 404 si no encuentra
    if (!$result) return $this->responseHttpNotFoundException($response);

    // Servicio Matomo
    /** @var MatomoService $matomo*/
    $matomo = $this->container->get('matomo');

    // Si no esta seteado Host Telus se consulta AddSite y setea el valor
    if(!$result['host_telus']) {
        $res = $matomo->addSite($result['host'], $result['host']);
        $db->update("website", [
            "host_telus" => $result['homepage']
        ], [
            'id' => $result['id']
        ]);
    }
    // Se pide el id y con el se pide el Script
    $id = $matomo->getSiteFromUrl($result['host_telus'])[0]['idsite'];
    $script = $matomo->getSiteScript($id)['value'];

    return $this->responseHttpOk($response, ["id" => $id,"script" => $script]);
}
```

Figura 4: Patrón Alta cohesión usado en la propuesta de solución. Método __invoke de la clase GetCode del paquete Telus.

(Fuente: Elaboración propia)

Experto:

Experto es un patrón que se usa más que cualquier otro al asignar responsabilidades; Es un principio básico que suele utilizarse en el diseño orientado a objetos. Con él no se pretende designar una idea oscura ni extraña; expresa simplemente la "intuición" de que los objetos hacen cosas relacionadas con la información que poseen. Ofrece una analogía con el mundo real. Acostumbramos asignar responsabilidad a individuos que disponen de la información necesaria para llevar a cabo una tarea (Larman 1999). En la propuesta de solución se utilizó este patrón en la clase SubscriptionApi, la cual representa una suscripción y cuenta con los datos propios de una, por lo que posee la información necesaria para calcular su precio.

Bajo acoplamiento:

El acoplamiento es una medida de la fuerza con que una clase está conectada a otras clases, con que las conoce y con que recurre a ellas. Una clase con bajo (o débil) acoplamiento no depende de muchas otras. El Bajo Acoplamiento es un principio que debemos recordar durante las decisiones de diseño: es la meta principal que es preciso tener presente siempre. Es un patrón evaluativo que el diseñador aplica al juzgar sus decisiones de diseño (Larman 1999). En la propuesta de solución se utilizó este patrón en clases como UserInfoApi o WebsiteApi las cuales representan a objetos del dominio del problema, también en clases controladoras como HomeNotifications o UserNotifications las cuales se relacionan con la menor cantidad de clases posibles para el cumplimiento de sus funciones.

Creador:

La creación de objetos es una de las actividades más frecuentes en un sistema orientado a objetos. En consecuencia, conviene contar con un principio general para asignar las responsabilidades concernientes a ella. El patrón Creador guía la asignación de responsabilidades relacionadas con la creación de objetos. El propósito fundamental de este patrón es encontrar un creador que debemos conectar con el objeto producido en cualquier evento (Larman 1999). En la propuesta de solución se utilizó este patrón en las diferentes clases controladoras encargadas de gestionar determinado recurso o información utilizando las clases representativas de cada uno como podrían ser SubscriptionApi. Las clases

controladoras tienen un método específico para la creación de los objetos representativos de los datos con los que operan, en la siguiente figura se muestra el método para la creación de PackageApi que utiliza el controlador mostrado en la figura 4.

```php
public function buildPackageApi($p_data): PackageApi
{
    $packageApi = new PackageApi();
    $packageApi->setId($p_data['id']);
    $packageApi->setName($p_data['name']);
    $packageApi->setDescription($p_data['description']);
    if($p_data['feature_pricing_tables']){
        $packageApi->setFeatures($p_data['feature_pricing_tables']);
    }
    $packageApi->setShow_in_home($p_data['show_in_home']);
    $packageApi->setShow_btn_home_quota($p_data['show_btn_home_quota']);
    $packageApi->setShow_btn_home_subscription($p_data['show_btn_home_subscription']);
    $packageApi->setShow_in_pricing_tables($p_data['show_in_pricing_tables']);

    return $packageApi;
}
```

Figura 5: Patrón Creador usado en la propuesta de solución. Método buildPackageApi de la clase Ge-tAll del paquete Packages.

(Fuente: Elaboración propia)

Patrones GOF

Hay varias categorías de patrones de diseño utilizados en la programación orientada a objetos, según el tipo de problema que aborden y/o los tipos de soluciones que nos ayuden construir. En su libro, Gang of Four presenta 23 patrones de diseño, divididos en tres categorías: creacional (tratan diferentes aspectos de la creación de objetos. Su objetivo es proporcionar mejores alternativas para situaciones en las que la creación directa de objetos no es conveniente), estructural (propone una forma de componer objetos para crear nuevas funcionalidades) y de comportamiento (se ocupan de los algoritmos y la asignación de responsabilidades entre objetos) (Ayeva y Kasampalis 2018).

Fábrica Abstracta (*Abstract factory*):

El patrón de Abstract factory (creacional) se usa normalmente cuando tenemos múltiples posibles implementaciones de un sistema que dependen de algún problema de configuración o plataforma. El código de llamada solicita un objeto de la fábrica abstracta, sin saber exactamente qué clase de objeto se devolverá. La implementación subyacente devuelta puede dependen de una variedad de factores, como la configuración regional actual, el sistema operativo o configuración (Phillips, Giridhar y Kasampalis 2016). En la propuesta de solución se evidencio la utilización de este patrón mediante la clase AppFactory perteneciente al framework slim cuya finalidad es la creación del objeto app, núcleo de la aplicación.

```
// Instantiate the app
$app = AppFactory::createFromContainer($container);
$app->setBasePath($settings['base_path']);
```

Figura 6: Patrón Abstract factory usado en la propuesta de solución. La clase AppFactory del frame-work Slim.

(Fuente: Elaboración propia)

Cadena de Responsabilidades (*Chain of Responsibility*)

Es un patrón de diseño que consta de una fuente de objetos y una serie de funciones o métodos. Cada funcione o métodos contiene lógica que define los tipos de objetos que puede manejar; el resto se pasa a la siguiente función en la cadena. También existe un mecanismo para agregar nuevos objetos de procesamiento al final de esta cadena (Phillips, Giridhar y Kasampalis 2016). En la propuesta de solución se evidencio la utilización de este patrón mediante los middlewares de procesamiento de solicitudes incorporados en el framework Slim además de los adicionados para la autenticación de usuarios y el analizador de solicitudes para la extracción de los datos enviados en el cuerpo de la misma.

II.4.3 Modelado de datos

El modelado de datos da la capacidad al ingeniero de software de representar objetos de datos, sus características y relaciones. Describe los elementos de la realidad que intervienen en

determinado problema para así crear una forma de representación de los mismos, es una descripción de una base de datos (Pressman 2010).

En el contexto de la situación problemática y en correspondencia con las características de la propuesta de solución existen varias fuentes de datos. La primera y más evidente es la base de datos de SeoWebMas, en un segundo lugar estarían otras fuentes de datos como el evaluador, el directorio de sitios web y Telus. La propuesta de solución hace uso integro de todas estas fuentes de datos y la ausencia de alguna de ellas puede causar fallas. El modo de acceso a las fuentes de datos es consulta directa a la base de datos de SeoWebMas y a través de servicios especializados para cada una de las demás fuentes.

Las principales tablas a consultar de la base de datos de SeoWebMas son:

- **webmaster:** Representa los usuarios del sistema.
- **subscription:** Contiene los datos de todas las subscripciones realizadas.
- **website:** Contiene los datos de los sitios web.
- **package:** Contiene los datos de los paquetes ofrecidos por el proyecto.
- **website_permission:** Representa la unión entre website, subscription, el webmaster y el package.
- **temporal_notification:** Contiene datos de las notificaciones públicas, emitidas todos los usuarios del sistema.
- **inbox_notification:** Contiene datos de las notificaciones privadas, emitidas usuarios específicos del sistema.

Los servicios consultados devuelven una respuesta en formato JSON, un ejemplo de la estructura de la respuesta recibida del evaluador al solicita los grupos de variables soportadas por el proyecto sería el siguiente:

```typescript
export interface Group {
    id:          number;
    name:        string;
    description: string;
    variables:   Variable[];
}

export interface Variable {
    complexity:  Complexity | null;
    impact:      Impact | null;
    slug:        string;
    informative: null;
    data:        null;
    name:        string;
    description: string;
    indicators:  Indicator[];
    group_id:    number;
}

export interface Complexity {
    id:         number;
    complexity: string;
}

export interface Impact {
    id:     number;
    impact: string;
}

export interface Indicator {
    slug: string;
    name: string;
}
```

Figura 7: Estructura de datos de respuesta al solicitar los grupos de variables al evaluador. (Fuente: Elaboración propia)

II.4.4 Diseño arquitectónico

La arquitectura de software de un programa o sistema de cómputo es la estructura o estructuras del sistema, lo que comprende a las componentes de software, sus propiedades externas visibles y las relaciones entre ellos. La arquitectura no es el software operativo, es una representación que permite analizar la efectividad del diseño para cumplir los requerimientos establecidos, considerar alternativas arquitectónicas en una etapa en la que hacer cambios al diseño todavía es relativamente fácil y reunir los riesgos asociados con la construcción del software (Pressman 2010). La propuesta de solución utiliza el patrón arquitectónico Modelo-Vista-Controlador (MVC) y el estilo REST (descrito en el epígrafe I.3). A continuación, se detallan elementos característicos del patrón arquitectónico MVC.

Patrón arquitectónico MVC

El patrón MVC propone la separación de la aplicación en tres partes esenciales, permitiendo que la misma pueda ser fácilmente modificada y mantenida. Las tres partes definidas por el patrón son el modelo, la vista y el controlador. Esas tres partes están interconectadas y se reparten las responsabilidades de mostrar, procesar y almacenar la información (Phillips,

Giridhar y Kasampalis 2016); el Modelo es la capa que contiene toda la información sobre los datos, la Controladora es la capa que se encarga de la lógica de negocio, cómo acceder a los datos y validarlos teniendo en cuenta su comportamiento y relaciones, la Vista es la capa que contiene la API REST (Deacon 2009).

En la figura 8 se muestran los principales elementos del patrón relacionados entre ellos y el papel de cada uno al momento de interactuar con los usuarios. En la propuesta de solución un evento podría iniciar con un usuario haciendo una consulta a cualquiera de los *endpoints* definidos, lo que haría que se llame a la clase controladora encargada de manejar ese evento, dicha clase cuenta con mecanismos para acceder a las entradas del usuario (en caso de haberla) y en función al tipo de solicitud crear, modificar, leer o eliminar recursos de la base de datos (representados en el modelo). El proceso debe terminar con la clase controladora enviando datos en formato JSON con la respuesta a la solicitud del usuario (la vista).

Utilizar el patrón arquitectónico MVC en conjunto con el estilo arquitectónico REST permite un amplio grado de desacople en la aplicación, facilitando aún más tanto el desarrollo de nuevas funcionalidades como la corrección de errores y mejorando la tolerancia a fallos de la aplicación. Esta mezcla de estilos también hace posible que los datos y servicios brindados por el sistema puedan ser consumidos por un sin número de clientes y en la forma que más conveniente les sea. La figura 9 muestra un diagrama donde se representa la combinación de estos dos estilos.

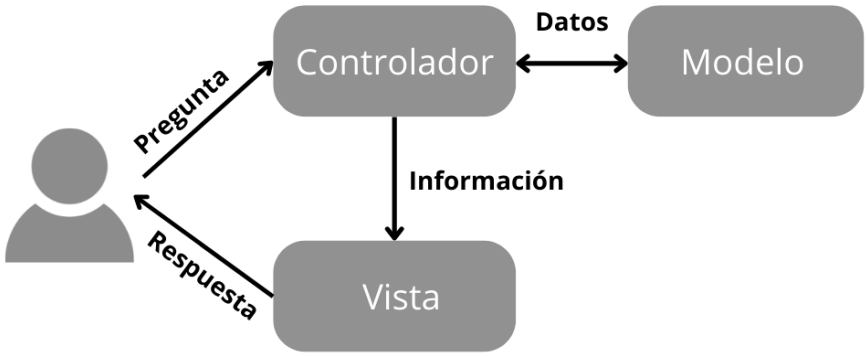

Figura 8: Patrón arquitectónico MVC

(Fuente: Elaboración Propia)

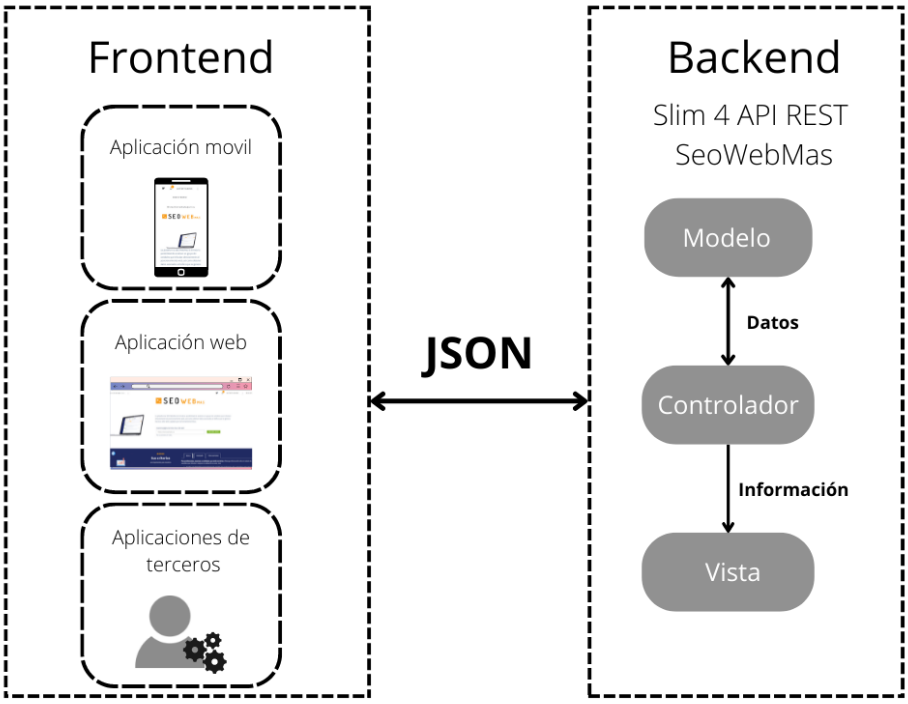

Figura 9: Arquitectura del sistema

(Fuente: Elaboración propia)

Para la representación de la arquitectura de la propuesta de solución se elaboró un diagrama de paquetes que contiene los elementos principales encargados de representar los

componentes del patrón MVC, sin entrar en muchos detalles. La figura 10 representa dicho diagrama.

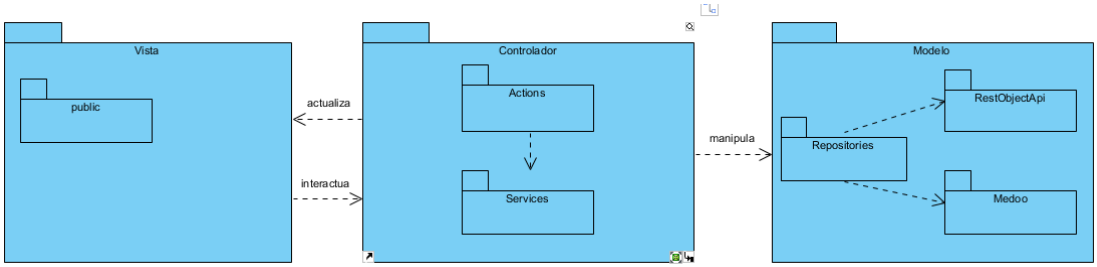

Figura 10: Diseño arquitectónico de la propuesta de solución.

(Fuente: Elaboración propia)

Descripción de la arquitectura de la aplicación

- **Vista:** El paquete vista contiene el subpaquete *public* donde se crea el objeto app, representativo de la API REST.
- **Controlador:** Contiene los subpaquetes *Actions* y *Services*, los cuales contienen las clases controladoras y las clases de acceso a servicios externos respectivamente.
- **Modelo:** Contiene los subpaquetes *RestObjectApi* y *Medoo*, los cuales contienen las clases que representan los datos dentro de la aplicación y el framework de comunicación con la base de datos.

Conclusiones del capítulo

La elaboración del modelo conceptual permitió caracterizar el contexto del negocio a informatizar para conocer las necesidades del cliente y los usuarios finales, así como las restricciones existentes en la propuesta de solución. Se determinaron 15 requisitos funcionales y 6 no funcionales. El diseño de las clases mediante la utilización de los patrones GRASP y GOF posibilitó la obtención de un diseño que facilita el mantenimiento de la propuesta de solución. El diseño arquitectónico basado en el patrón MVC y el estilo REST facilitó la comunicación entre los diferentes componentes de la API REST implementada.

Capítulo III: Implementación y evaluación de la API REST para la gestión de los datos y servicios brindados desde el Monitor de sitios Web

En el presente capítulo se abordan dos de las etapas más importantes del desarrollo de software, la etapa de implementación donde se definen y se organiza el código que solventará la problemática abordada en el presente trabajo de diploma de acuerdo con las tecnologías definidas en el epígrafe 1.8. Además, se materializarán los requisitos de software y la arquitectura definida en el capítulo anterior en forma de código, quedando conformado el producto requerido por el cliente.

La segunda etapa abordada en este capítulo es la de validación o evaluación del sistema, en ella se pondrán a prueba diversos aspectos del sistema para garantizar que se cumplan con los estándares requeridos por el cliente y el correcto funcionamiento de las tareas que serán llevada a cabo por el sistema. Para alcanzar el objetivo de esta fase de desarrollo de la propuesta de solución, el producto obtenido en la fase de implementación será sometido a varias pruebas e iteraciones donde se rectificarán las fayas encontradas hasta que se tenga certeza de la fiabilidad del sistema.

III.1 Diagrama de componentes

Un diagrama de componentes es la representación de los componentes del sistema, interfaces y relaciones. Un componente es la implementación física de un conjunto de otros elementos lógicos, como clases y colaboraciones (Pressman 2010).

La siguiente figura muestra el diagrama de componentes de la propuesta de solución. Se puede apreciar la estructura de los paquetes y la distribución de los componentes dentro de estos. El diagrama es solo una representación de alto nivel del sistema, paquetes que son declarados, pero por su complejidad no pueden ser especificados dentro del diagrama.

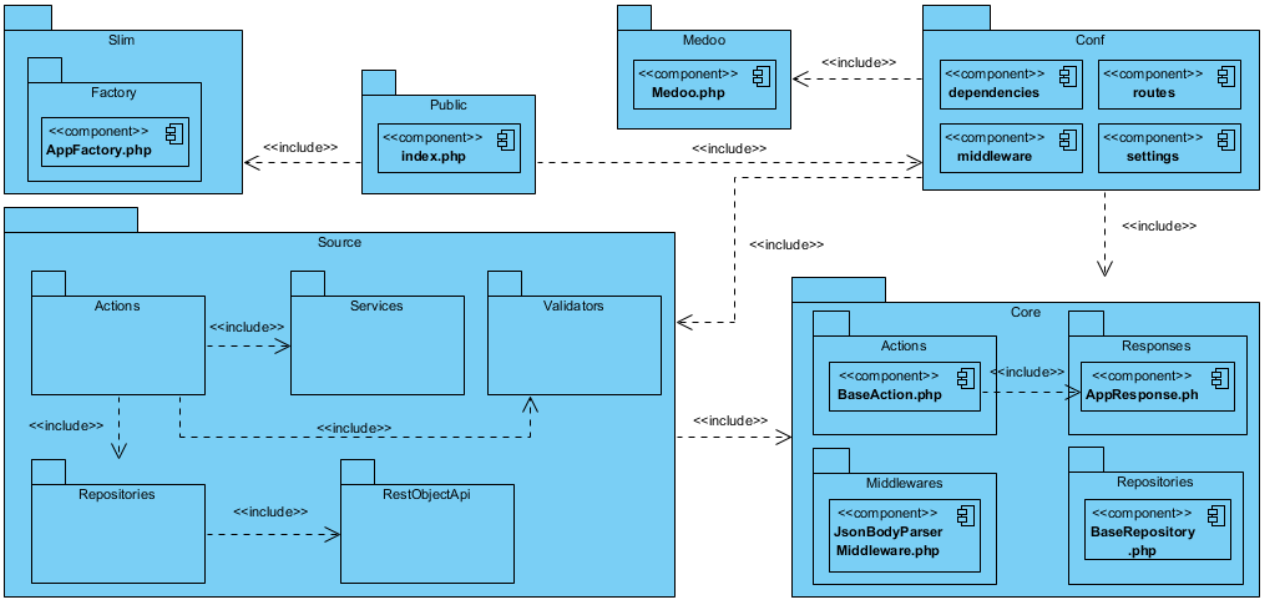

Figura 11: Diagrama de componentes de la propuesta de solución.
(Fuente: Elaboración propia)

El diagrama anterior representa la estructura del proyecto de la propuesta de solución, el punto de entrada sería el archivo index.php que se encuentra en el paquete Public. En cada solicitud que se realice al sistema se crearía una instancia de la aplicación de Slim, se configuraría a través de las clases del paquete Conf, donde se definen dependencias como Medoo y se crean la rutas que tendrá el sistema. Cada una de las rutas estará en correspondencia con una clase dentro del paquete Actions, encargada de implementar la lógica necesaria para satisfacer un requisito de software. Las clases Actions reciben datos a través de los repositorios y los servicios y validan los datos de entrada de los usuarios mediante las clases validadoras del paquete Validators. En la propuesta de solución se utilizaron las relaciones entre clases del tipo herencia como buena práctica, por lo que cada clase del paquete Source está basada en otra del paquete Core.

III.2 Estándares de codificación utilizados

Según fuentes como (Hiken 2020) los estándares de codificación mejoran la seguridad y la protección. El objetivo de los estándares de codificación de software es inculcar prácticas de

programación probadas que conduzcan a un código seguro, confiable, comprobable y mantenible. Por lo general, esto significa evitar las prácticas de codificación inseguras conocidas o el código que puede causar un comportamiento impredecible. Los estándares permiten un estilo de programación homogéneo, fácilmente entendible por todos los participantes del equipo de desarrollo. El código de la propuesta de solución fue escrito en inglés y utilizando los siguientes estándares:

- Indentación: Para la Indentación se utilizó 1 tabulador por cada nivel.
- Nomenclatura de las funciones: Los nombres de funciones deberán estar escritos en letras minúsculas, en el caso de que sea una palabra compuesta empezará con mayúscula en la segunda palabra.
- Nomenclatura de las clases: Los nombres de las clases serán con mayúscula, en caso de ser un nombre compuesto las siguientes palabras se escribirán de igual forma.
- Nomenclatura de las variables: Los identificadores para las variables y los parámetros serán con letras en minúsculas y en caso de ser un nombre compuesto las siguientes palabras se escribirán con mayúscula.
- Los nombres de variables o funciones deben ser lo suficientemente descriptivos, sin exceder de 30 caracteres.
- Para la implementación de los controladores se utilizarán clases invocables, en el contexto de *Slim framework* son clases que poseen un método llamado __invoke donde se lleva a cabo la lógica del controlador.
- Todas las funciones deben tener comentarios, que describan su propósito y declaren definición de la especificación Open Api.
- Para mejorar el autocompletado de algunos editores de código, principalmente el seleccionado en el apartado de tecnologías se empleará phpDocs (comentarios con una estructura específica sobre las variables o funciones en cuestión que aporta información sobre el tipo de variable o los parámetros recibidos por funciones, así como el valor que devuelve).

```php
 * Return all the packages offered by the project
 * @param Request $request
 * @param Response $response
 * @return Response
 * @throws Exception
 */
public function __invoke(Request $request, Response $response): Response
{
    /** @var Medoo $db */
    $db = $this->container->get('em');

    $sql = 'select id, name, description, feature_pricing_tables, show_in_home,
            show_btn_home_quota, show_btn_home_subscription, show_in_pricing_tables
            from package;';

    $result = $db->query($sql)->fetchAll();
    $data = array();
    if (count($result)) {
        foreach ($result as $p) {
            $data[] = $this->buildPackageApi($p);
        }
    }
    return $this->responseHttpOk($response, $data);
}
```

Figura 12: Ejemplo de función que cumple con todos los estándares de codificación antes listados.

(Fuente: Elaboración propia)

III.3 Diagrama de despliegue

El diagrama de despliegue permite modelar la disposición física de un sistema. Muestra el hardware usado y los componentes instalados en el hardware, además de las conexiones físicas entre este y las relaciones entre componentes. Es utilizado para capturar los elementos de configuración del procesamiento y las conexiones entre esos elementos. También se utiliza para visualizar la distribución de los componentes de software en los nodos físicos (Muñoz González 2018). La siguiente figura muestra el diagrama de despliegue de la propuesta de solución.

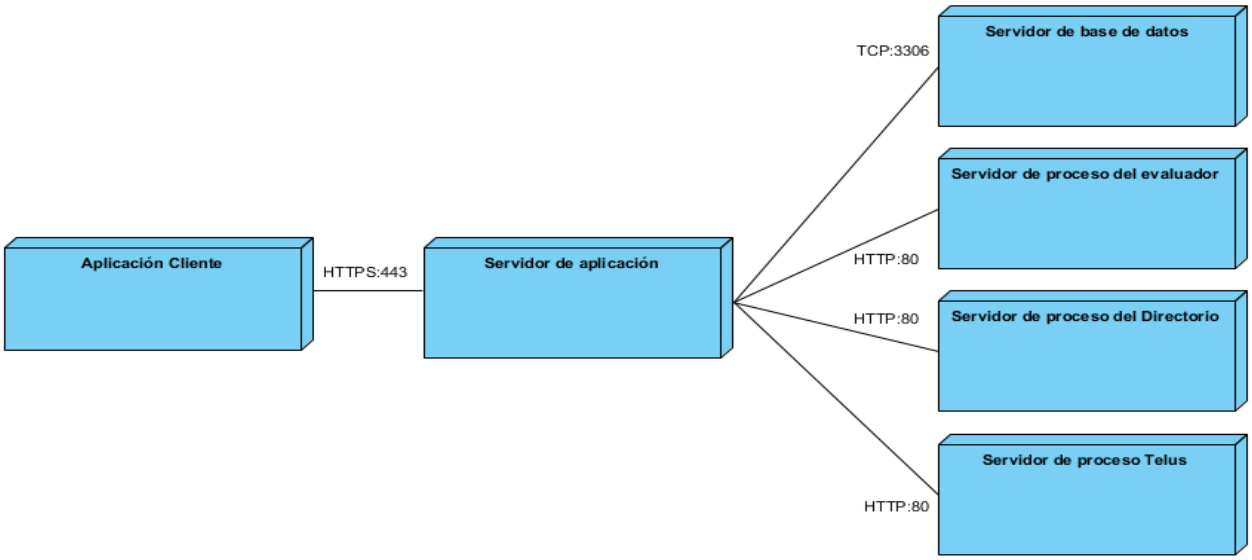

Figura 13: Diagrama de despliegue de la propuesta de solución.

(Fuente: Elaboración propia)

Descripción de los nodos del diagrama de despliegue

- Aplicación Cliente: representa cualquier dispositivo que ejecute alguna aplicación consumidora de los servicios ofrecidos por la propuesta de solución.
- Servidor de aplicación: contendrá la API REST implementada.
- Servidor de base de datos: contiene un servicio de MySQL ejecutándose donde se guardan los datos de SeoWebMas.
- Servidor de proceso del evaluador: contiene el servicio del evaluador en ejecución.
- Servidor de proceso del Directorio: contiene el servicio del Directorio en ejecución.
- Servidor de proceso Telus: contiene el servicio de Telus en ejecución.

El diagrama de despliegue presentado plantea que la comunicación entre el usuario final o cliente y la propuesta de solución se realice mediante el protocolo HTTPS. A su vez la propuesta de solución se estaría comunicando con los servicios del centro mediante el protocolo HTTP y con la base de datos de SeoWebMas mediante el protocolo TCP y puerto 3306.

III.4 Pruebas de software para la evaluación de la propuesta de solución

Las pruebas de software son necesarias para descubrir errores que se cometieron de forma inadvertida conforme se diseñó y construyó el software. Con frecuencia las pruebas requieren más esfuerzo que cualquier otra acción de ingeniería del software. Si se realizan sin orden se desprecia tiempo, se emplea esfuerzo innecesario y es posible que pasen errores desapercibidos, por lo que es razonable establecer una estrategia sistémica para probar el software. Una prueba de software debe incluir pruebas de bajo nivel que son necesarias para verificar que un pequeño segmento de código funcione, así como pruebas de alto nivel, que validan las principales funciones del sistema a partir de los requerimientos del cliente (Pressman 2010).

La metodología de desarrollo de software AUP para la UCI establece 3 disciplinas para la ejecución de pruebas: internas, liberación y aceptación. Como parte de las pruebas se decide la realización de las pruebas internas y de aceptación, se descartan las pruebas de liberación ya que estas son diseñadas y ejecutadas por una entidad certificadora de la calidad externa. En las pruebas internas se propone verificar el resultado de la aplicación. En los siguientes subepígrafes se describen las pruebas de software aplicadas, métodos y técnicas utilizadas en la evaluación de la propuesta de solución.

III.4.1 Pruebas internas

En esta disciplina se verifica el resultado de la implementación probando cada construcción, incluyendo tanto las construcciones internas como intermedias, así como las versiones finales a ser liberadas (Rodríguez Sánchez 2015). Para la evaluación de la propuesta de solución se aplicaron pruebas unitarias mediante el método de caja blanca y la técnica de camino básico en conjunto con pruebas funcionales mediante el método de caja negra y la técnica de partición de equivalencia.

Pruebas unitarias

Las pruebas unitarias se centran en probar cada componente de código de un software de forma individual para asegurar que funcione de manera apropiada como unidad. Emplean

técnicas de prueba que recorren caminos específicos en la estructura de control de los componentes (pruebas estructurales) (Pressman 2010). El método de prueba y la técnica seleccionada para realizar este tipo de prueba son explicados a continuación.

Método de prueba: Caja blanca

La prueba de caja blanca del software se basa en el examen cercano de los detalles de procedimientos. Las rutas lógicas a través del software y las colaboraciones entre componentes se ponen a prueba al revisar conjuntos específicos de condiciones y/o bucles. Para realizar una prueba de caja blanca lo único que se necesita es definir todas las rutas lógicas, desarrollar casos de prueba para revisarlas y evaluar resultados, generando así casos de prueba para revisar de manera exhaustiva la lógica del programa (Pressman 2010).

Técnica de prueba: Camino básico

La técnica del Camino básico permite obtener una medida de la complejidad lógica de la codificación de software y usar esa medida como guía para la definición de un conjunto básico de caminos de ejecución independiente en un componente o programa. Un camino o ruta es una vía por la cual procede la ejecución a través de una función desde su inicio hasta el fin (Pressman 2010). A continuación, se describen los pasos llevados a cabo al aplicar la técnica del camino básico a la función que satisface el RF. 13 “Mostrar código de Telus”.

Aplicación de la técnica del camino básico

1. Dibujo del grafo de flujo de la funcionalidad o procedimiento a analizar.

```php
public function __invoke(Request $request, Response $response, array $args): Response
{
    /** @var Medoo $db*/
    $db = $this->container->get('em');

    // Seleccionar los atributos del sitio pasado por los argumentos, el sitio debe ser del usuario autenticado.
    $sql = 'Select w.id, w.host, w.homepage, w.host_telus
            from website w join website_permission wp on w.id = wp.website_id
            where w.id = \'' . $args['website'] . '\'
            and wp.owner = \'' . $this->getUserNameJwt($request) . '\'
            ';
    $result = $db->query($sql)->fetch();

    // Retornar 404 si no encuentra
    if (!$result)
        return $this->responseHttpNotFoundException($response);

    // Servicio Matomo
    /** @var MatomoService $matomo*/
    $matomo = $this->container->get('matomo');

    // Si no esta seteado Host Telus se consulta AddSite y setea el valor
    if(!$result['host_telus']) {
        $res = $matomo->addSite($result['host'], $result['host']);
        $db->update("website", [
            "host_telus" => $result['homepage']
        ], [
            'id' => $result['id']
        ]);
    }
    // Se pide el id y con el se pide el Script
    $id = $matomo->getSiteFromUrl($result['host_telus'])[0]['idsite'];
    $script = $matomo->getSiteScript($id)['value'];

    return $this->responseHttpOk($response, [
        "id" => $id,
        "script" => $script
    ]);
}
```

Figura 14: Código que satisface el RF. 13 "Mostrar código de Telus"

(Fuente: Elaboración propia)

El grafo de flujo del código mostrado en la figura 14 es el siguiente:

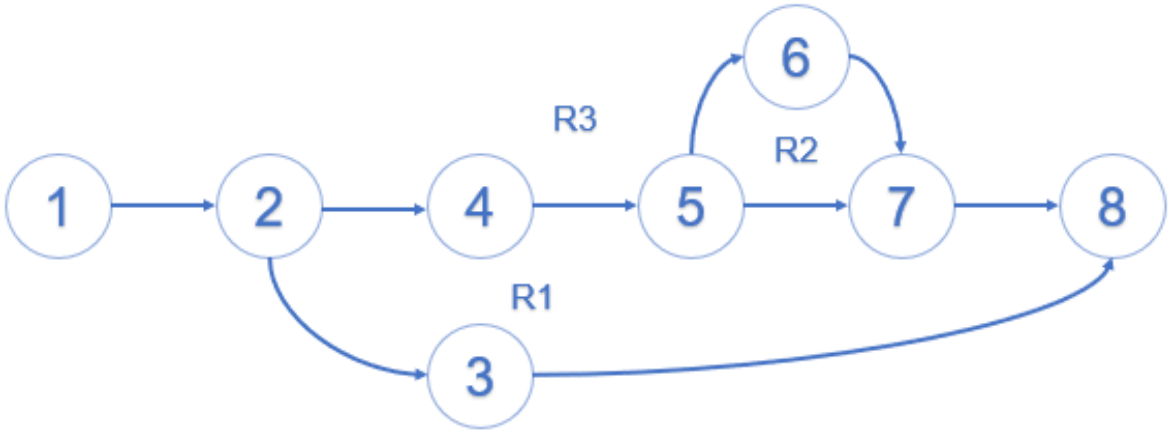

Figura 15: Grafo de flujo del código que satisface el RF. 13 "Mostrar código de Telus"

(Fuente: Elaboración propia)

2. Cálculo de la complejidad ciclomática del grafo de flujo de la implementación del RF. 13 "Mostrar código de Telus"

La complejidad ciclomática es una mediación de software que proporciona una evaluación cuantitativa de la complejidad lógica de un programa. En el contexto del método de prueba de ruta básica, su valor representa el número de rutas independientes del conjunto básico de un programa, brindando así una cota superior para el número de pruebas que se deben realizar para asegurar que todos los enunciados se ejecuten al menos una vez (Pressman 2010).

Existen tres formas para calcular la complejidad ciclomática del grafo anterior (Pressman 2010):

- V(G) = A - N + 2
    - Donde A: número de aristas del grafo de flujo, N: número de nodos del grafo

  V(G) = 9 – 8 + 2 = 3

- V(G) = P + 1
    - P: número de nodos predicados (nodos con más de una arista de salida) contenidos en el grafo

  V(G) = 2 + 1 = 3

- V(G) = Regiones (R)
    - RN: son las áreas delimitadas por nodos y aristas en el grafo

  V(G) = 3

3. Determinar el conjunto básico de caminos linealmente independientes

   Camino básico 1: 1, 2, 4, 5, 7, 8

   Camino básico 2: 1, 2, 3, 8

   Camino básico 3: 1, 2, 4, 5, 6, 7, 8

4. Con el conjunto de caminos básicos se definen los casos de pruebas a ejecutar para garantizar que todas las sentencias del código se ejecuten al menos una vez, el proceso se documenta definiendo los siguientes elementos para cada caso de prueba:

- Descripción: contiene una descripción sobre las restricciones de los datos de entrada que debe tener el caso de prueba.
- Condición de ejecución: se especifican los parámetros que debe poseer el caso de prueba para que se cumpla una condición deseada como respuesta del funcionamiento del procedimiento.
- Entrada: se muestran los parámetros de entrada al procedimiento.
- Resultados esperados: se explica el resultado esperado de la ejecución del procedimiento.

Un ejemplo de un caso de prueba para el RF. 13 “Mostrar código de Telus” para el camino 3 donde se prueba la obtención satisfactoria del código de Telus para un sitio web que no está añadido al servicio de Telus.

Tabla 7: Diseño de casos de prueba para el camino 3 del RF. 13

Diseño de caso de prueba para el camino 3	
Descripción	Muestra el id de Telus y el script de un sitio web que no estaba añadido previamente al servicio de Telus.
Condición de ejecución	Se pasa por parámetro el id del sitio a consultar.
Entrada	website: “68af802b16a1202c3ab56c4472108c4fa573b3b5be26d1032367c15a74a1ce96”
Resultados esperados	Añadir el sitio al servicio de Telus, agregar el atributo *host_telus* a la tabla *website* de la base de datos del sitio pasado por parámetros y mostrar el id y script del sitio.

Al aplicar los casos de pruebas se comprobó que el flujo de trabajo de las funcionalidades es correcto, ya que cada sentencia fue ejecutada al menos una vez, cumpliéndose así las condiciones de las pruebas y los resultados esperados. Fueron necesarias 4 iteraciones de pruebas unitarias logrando solventarse 21 errores de funcionalidad y de comunicación entre procesos. La siguiente tabla muestra el resumen del proceso de detección y solución de errores según las iteraciones realizadas.

Tabla 8: Cantidad de no conformidades detectadas en el proceso de pruebas unitarias

No conformidades	Iteración 1	Iteración 2	Iteración 3	Iteración 4
Detectadas	11	6	4	0
Resueltas	11	6	4	0

Pruebas funcionales

Las pruebas funcionales son aquellas que se centran en los requerimientos del software con el objetivo de ejercitar profundamente el sistema comprobando la integración del sistema globalmente, verificando el funcionamiento correcto de las interfaces entre los distintos subsistemas que lo componen y con el resto de sistemas de información con los que se comunica (Pressman 2010). Para la realización de este tipo de prueba se seleccionó el método de Caja negra específicamente la técnica de Partición de equivalencia.

Método de prueba: Caja negra

También llamadas pruebas de comportamiento se enfocan en los requerimientos funcionales del sistema, permiten derivar conjunto de condiciones de entrada que revisaran por completo todos los requerimientos funcionales de un programa. Son un complemento a las técnicas de caja blanca, siendo muy probable que descubran una clase de error diferentes a los descubiertos por los métodos de caja blanca (Pressman 2010).

Técnica de prueba: Partición de equivalencia

La técnica divide el dominio de entrada de un programa en clases de datos, a partir de las cuales pueden derivarse casos de prueba. Un caso de prueba ideal descubre clases de errores, que, de otra manera, requeriría la ejecución de muchos casos antes de que se observe el error general. El diseño de casos de prueba para la partición de equivalencia se basa en una evaluación de las clases de equivalencia para una condición de entrada, donde las clases de equivalencia son un conjunto de estados validos o inválidos para las condiciones de entrada (Pressman 2010).

Diseños de caso de pruebas

Es una parte de las pruebas de componentes y sistemas en las se diseñan los casos de prueba (entradas y salidas esperadas) para probar el sistema. Su objetivo es crear un conjunto de casos de prueba que sean efectivos descubriendo defectos en los programas y muestren que el sistema satisface sus requerimientos. Para su diseño, se selecciona una característica del sistema o componente que se está probando, un conjunto de entradas que ejecutan dicha característica, se documentan las salidas esperadas o rasgos de salida y donde sea posible se diseña una prueba automatizada que demuestre que las salidas reales y las esperadas son las mismas (Muñoz González 2018). El caso de prueba del RF. 11 “Compartir sitio web” es el siguiente:

Abreviaturas utilizadas:

- CP: Caso de Prueba
- HU: Historia de Usuario
- EC: Escenario

Tabla 9: Caso de prueba funcional HU10: Compartir sitio web

Caso de prueba funcional	
CP1_HU11	HU10: Compartir sitio web
Responsable: Luis Enrique Reyes Pérez	
Descripción: El caso de prueba inicia cuando el usuario hace una petición POST a la URL asociada al requisito. Al ser una petición POST los datos son proporcionados en el cuerpo de la misma, en formato JSON. Los datos requeridos son: site_id, email.	

Tabla 10: Escenarios de caso de prueba funcional HU10 Compartir sitio web

Escenario	**Descripción**	**Respuesta del sistema**	**Flujo del sistema**

EC 1.1 Se introducen datos correctos	El usuario hace una petición POST proporcionando los datos requeridos en el cuerpo de la petición.	El sistema comparte el sitio con el usuario indicado, notificando a ambos del éxito de la petición.	El usuario introduce el id del sitio a compartir y el correo del usuario que lo recibe.
EC 1.2 Se introducen datos incorrectos	El usuario hace una petición POST proporcionando datos incorrectos en el cuerpo de la petición.	El sistema responde con un código de error 400 *Bad Request* en caso de tener un correo invalido y 404 *Not Found* en caso de que el sitio especificado no exista.	El usuario introduce el id del sitio a compartir y el correo del usuario que lo recibe.
EC 1.3 Se introducen datos vacíos	El usuario hace una petición POST proporcionando datos vacíos en el cuerpo de la petición.	El sistema responde con un código de error 400 *Bad Request.*	El usuario introduce el id del sitio a compartir y el correo del usuario que lo recibe como datos en blanco “ ”.
EC 1.4 No se introducen datos	El usuario hace una petición POST, pero no proporciona los datos en el cuerpo de la petición.	El sistema responde con un código de error 400 *Bad Request.*	El usuario hace la petición con el cuerpo vacío.

Para lograr el correcto funcionamiento del sistema fue necesario realizar 3 iteraciones de pruebas funcionales logrando solventarse 13 errores que evidenciaban problemas de validación de datos de entrada, errores de funcionalidad y sintácticos. La siguiente tabla

muestra el resumen del proceso de detección y solución de errores según las iteraciones realizadas.

Tabla 11: Cantidad de no conformidades detectadas en el proceso de pruebas

No conformidades	**Iteración 1**	**Iteración 2**	**Iteración 3**
Detectadas	9	4	0
Resueltas	9	4	0

III.4.2 Pruebas de integración

Las pruebas de integración son una forma de comprobar la correcta interrelación de los distintos componentes del sistema. El objetivo es tomar los componentes probados de manera individual y construir una estructura de programa que se haya dictado por diseño (Pressman 2010). En el caso de la solución desarrollada, es la verificación de una correcta interoperabilidad entre el sistema desarrollado y el ecosistema de aplicaciones del Monitor de Sitios Web con los que este estaría interactuando, validando la compatibilidad y el funcionamiento de las diferentes partes que componen la solución informática propuesta.

Para la realización de las pruebas de integración se llevaron a cabo diferentes acciones, a continuación, se mencionan las fundamentales:

- Integración entre la API REST desarrollada con los diferentes servicios con los que estaría interactuando.
- Integración de la API REST con las aplicaciones de monitor que consumirían sus datos.
- Verificación de la integración realizada y corrección de errores.

Tabla 12: Cantidad de no conformidades por cada iteración de las pruebas de integración.

No Conformidades	**Iteración 1**	**Iteración 2**	**Iteración 3**
Detectadas	6	4	0
Resueltas	6	4	0

La ejecución de las pruebas de integración permitió verificar el trabajo conjunto de los componentes de la propuesta de solución y el resto de las aplicaciones del Monitor. La gran mayoría de errores ocurrieron al integrar los datos de Telus con la propuesta de solución, evidenciándose respuestas erróneas por parte del sistema, también fueron solventadas otras no conformidades con lo que respecta al servicio del evaluador y el directorio. Se lograron detectar y corregir un total 10 errores y como resultado se verificó la correcta integración de la aplicación con los servicios y demás aplicaciones del Monitor de Sitios Web.

III.4.3 Pruebas de aceptación

Las pruebas de aceptación se realizan sobre el producto terminado e integrado; están concebidas para que sea un usuario final quien detecte los posibles errores y verifique el cumplimiento de las funcionalidades del software. Estas pruebas generalmente son funcionales y se basan en los requisitos definidos por el cliente. Se clasifican en dos tipos: pruebas Alfa y pruebas Beta. Las pruebas Alfa se realizan por un cliente en un entorno controlado por el equipo de desarrollo, y las pruebas Beta se realizan en las instalaciones propias de los clientes (Suárez y Fontela 2003). El cliente realizó pruebas de aceptación de tipo Alfa, en las cuales se detectaron 4 no conformidades y luego de 2 iteraciones de pruebas fueron solventadas. En conformidad con la herramienta desarrollada emitió un acta de aceptación de productos de trabajo.

Conclusiones del capítulo

Haber desarrollado un diagrama de componentes permitió conocer la representación de los componentes del sistema, interfaces y relaciones que serán implementadas para dar solución al problema. La definición de estándares de codificación permitió esclarecer y definir las mejores prácticas de programación que serán utilizadas en el desarrollo de la propuesta de solución. El diagrama de despliegue elaborado permitió tener una visión de la infraestructura necesaria para el despliegue de la API REST. La elaboración de pruebas le permitió al cliente evaluar el cumplimiento de los requisitos funcionales, evidenciándose una alta satisfacción del cliente hacia la solución desarrollada mediante pruebas Alfa.

Conclusiones finales

Una vez finalizada la presente investigación y cumplidos todos los objetivos específicos y tareas de investigación se llegó a las siguientes conclusiones:

- El estudio realizado sobre los fundamentos teóricos-metodológicos del desarrollo de interfaces de programación de aplicaciones permitió establecer el estilo API REST como el seleccionado para desarrollar la propuesta de solución, sus características principales y buenas prácticas de diseño.
- El estudio de homólogos realizado posibilitó validar el uso de las APIs para lograr la interoperabilidad entre aplicaciones, así como resaltar al estilo API REST como una de las alternativas más utilizadas.
- La implementación de la propuesta de solución posibilitó el desarrollo de una API REST para la integración de los datos dentro de las diferentes aplicaciones del Monitor de Sitios Web de forma controlada, segura y documentada.
- Las pruebas de software realizadas permitieron evaluar el funcionamiento de la API REST, la conformidad con los requisitos definidos y la eliminación de las no conformidades encontradas en las iteraciones, validando de esta forma el correcto funcionamiento de la propuesta de solución.

Recomendaciones

Se recomienda seguir integrando nuevos datos y servicios del Monitor de Sitios Web mediante la propuesta de solución. Además, del empleo de la solución implementada en el proceso de integración de datos tanto en aplicaciones del Monitor como de terceros para un acceso seguro, rápido y controlado a la información que ahí se maneja. Por otra parte, se aconseja delegar la responsabilidad del envío de correos a una cola de mensajería para conseguir que el cliente obtenga la respuesta del sistema lo más rápido posible.

Referencias bibliográficas

Amazon S3. *ProgrammableWeb* [en línea], 2022. [Consulta: 9 junio 2022]. Disponible en: https://www.programmableweb.com/api/amazon-s3.

AMBLER, S.W., 2017. Simple Tools for Software Modeling -OR- It's «Use the Simplest Tool» not «Use Simple Tools». [en línea]. [Consulta: 7 junio 2022]. Disponible en: http://www.agilemodeling.com/essays/simpleTools.htm#SelectingCASE.

ARNAUD, L., 2019. *The Design of Web APIs*. Shelter Island, NY 11964: Manning Publications Co. ISBN 978-1-61729-510-2.

AUTH0.COM, 2022. JWT.IO - JSON Web Tokens Introduction. [en línea]. [Consulta: 8 septiembre 2022]. Disponible en: http://jwt.io/.

AYEVA, K. y KASAMPALIS, S., 2018. *Mastering Python Design Patterns*. Second Edition. S.l.: Packt Publishing Ltd. ISBN Packt Publishing Ltd.

BARBAR, A. y ISMAIL, A., 2019. Search Engine Optimization (SEO) for Websites. *Proceedings of the 2019 5th International Conference on Computer and Technology Applications* [en línea]. New York, NY, USA: Association for Computing Machinery, pp. 51-55. [Consulta: 10 mayo 2022]. ISBN 978-1-4503-7181-0. DOI 10.1145/3323933.3324072. Disponible en: https://doi.org/10.1145/3323933.3324072.

Braintree GraphQL API. [en línea], 2022. [Consulta: 8 octubre 2022]. Disponible en: https://graphql.braintreepayments.com/.

CLEVELAND, S.B., JAMTHE, A., PADHY, S., STUBBS, J., PACKARD, M., LOONEY, J., TERRY, S., CARDONE, R., DAHAN, M. y JACOBS, G.A., 2020. Tapis API Development with Python: Best Practices In Scientific REST API Implementation: Experience implementing a distributed Stream API. *Practice and Experience in Advanced Research Computing* [en línea]. New York, NY, USA: Association for Computing Machinery, pp. 181-187. [Consulta: 14 mayo 2022]. ISBN 978-1-4503-6689-2. Disponible en: https://doi.org/10.1145/3311790.3396647.

Códigos de estado de respuesta HTTP - HTTP | MDN. [en línea], 2022. [Consulta: 4 junio 2022]. Disponible en: https://developer.mozilla.org/es/docs/Web/HTTP/Status.

DEACON, J., 2009. Model-View-Controller (MVC) Architecture. ,

Digital Report 2022: El informe sobre las tendencias digitales, redes sociales y mobile. *We Are Social Spain* [en línea], 2022. [Consulta: 28 junio 2022]. Disponible en: https://wearesocial.com/es/blog/2022/01/digital-report-2022-el-informe-sobre-las-tendencias-digitales-redes-sociales-y-mobile/.

DOGLIO, F., 2018. *REST API Development with Node.js* [en línea]. S.l.: s.n. [Consulta: 19 mayo 2022]. ISBN 978-1-4842-3714-4. Disponible en: https://link.springer.com/book/10.1007/978-1-4842-3715-1.

EBNER, T. y GRANITZA, L., 2019. *The SEO Book. The way to Nr. 1*. Germany, L-2338, Luxembourg: EDISON VERLAG.

Editores de código. [en línea], 2022. [Consulta: 9 junio 2022]. Disponible en: https://desarrolloweb.com/colecciones/editores-codigo.

FREED, N., KLENSIN, J.C. y HANSEN, T., 2013. Media Type Specifications and Registration Procedures. [en línea]. Request for Comments. S.l.: Internet Engineering Task Force. [Consulta: 9 junio 2022]. RFC 6838. Disponible en: https://datatracker.ietf.org/doc/rfc6838.

GEEWAX, J., 2021. *API Design Patterns*. Shelter Island, NY 11964: Manning Publications Co. ISBN 978-1-61729-585-0.

GraphQL | A query language for your API. [en línea], 2022. [Consulta: 4 junio 2022]. Disponible en: https://graphql.org/.

GUERRA, C.A., 2017. Obtención de Requerimientos. Técnicas y Estrategia. *SG Buzz* [en línea]. [Consulta: 18 julio 2022]. Disponible en: https://sg.com.mx/revista/17/obtencion-requerimientos-tecnicas-y-estrategia.

HIKEN, A., 2020. Estándares de codificación de software y pautas de programación. [en línea]. [Consulta: 31 agosto 2022]. Disponible en: https://es.parasoft.com/blog/an-ounce-of-prevention-software-safety-security-through-coding-standards/.

HUSSEIN, S., 2021. Review of Web Service Technologies: REST over SOAP. , pp. 2020. ISSN 2521-3504. DOI 10.29304/jqcm.2020.12.4.715.

IBM Docs - RPC. [en línea], 2022. [Consulta: 19 mayo 2022]. Disponible en: https://prod.ibmdocs-production-dal-6099123ce774e592a519d7c33db8265e-0000.us-south.containers.appdomain.cloud/docs/en/aix/7.1?topic=concepts-remote-procedure-call.

JOSKOWICZ, J., 2008. *Reglas y Prácticas en eXtreme Programming*. S.l.: s.n.

LARMAN, C., 1999. *UML y Patrones. Introducción al análisis y diseño orientado a objetos*. PRENTINCE HALL, México: s.n. ISBN 970-17-0261-1.

LEE, Y. y LIU, Y., 2022. Using refactoring to migrate REST applications to gRPC. *Proceedings of the 2022 ACM Southeast Conference* [en línea]. New York, NY, USA: Association for Computing Machinery, pp. 219-223. [Consulta: 19 mayo 2022]. ISBN 978-1-

4503-8697-5. DOI 10.1145/3476883.3520220. Disponible en: https://doi.org/10.1145/3476883.3520220.

LEWANDOWSKI, D., SÜNKLER, S. y YAGCI, N., 2021. The influence of search engine optimization on Google's results: A multi-dimensional approach for detecting SEO. *13th ACM Web Science Conference 2021* [en línea]. New York, NY, USA: Association for Computing Machinery, pp. 12-20. [Consulta: 10 mayo 2022]. ISBN 978-1-4503-8330-1. DOI 10.1145/3447535.3462479. Disponible en: https://doi.org/10.1145/3447535.3462479.

LinkedIn. *ProgrammableWeb* [en línea], 2022. [Consulta: 9 junio 2022]. Disponible en: https://www.programmableweb.com/api/linkedin.

MENA ROA, M., 2021. • Gráfico: ¿Cuántos sitios web hay en el mundo? | Statista. [en línea]. [Consulta: 28 junio 2022]. Disponible en: https://es.statista.com/grafico/19107/numero-de-sitios-web-existentes-en-internet/.

Most Popular API Authentication Methods | 3Pillar Global. [en línea], [sin fecha]. [Consulta: 8 septiembre 2022]. Disponible en: https://www.3pillarglobal.com/insights/most-popular-api-authentication-methods/.

MUÑOZ GONZÁLEZ, D. de la C., 2018. *API REST asociada al repositorio de Nova*. LA HABANA: s.n.

MURPHY, G., 2018. *The power of PHP*. LLC 243 5th Avenue, Suite 136, New York: Cavendish Square Publishing. First Edition. ISBN 978-1-5026-2946-3.

NETFLIX TECHNOLOGY, 2021. Practical API Design at Netflix, Part 1: Using Protobuf FieldMask. *Medium* [en línea]. [Consulta: 8 octubre 2022]. Disponible en: https://netflixtechblog.com/practical-api-design-at-netflix-part-1-using-protobuf-fieldmask-35cfdc606518.

OpenAPI Specification. [en línea], 2022. [Consulta: 9 junio 2022]. Disponible en: https://swagger.io/specification/.

PÉREZ PORTO, J. y MERINO, M., 2012. Definición de lenguaje de programación — Definicion.de. *Definición.de* [en línea]. [Consulta: 7 junio 2022]. Disponible en: https://definicion.de/lenguaje-de-programacion/.

PHILLIPS, D., GIRIDHAR, C. y KASAMPALIS, S., 2016. *Python: Master the Art of Design Patterns*. S.l.: Packt Publishing Ltd. ISBN 978-1-78712-518-6.

PRESSMAN, R.S., 2010. *Ingienería de Software. Un enfoque práctico.* séptima edición. S.l.: s.n. ISBN 978-607-15-0314-5.

¿Qué es Apache y para qué sirve? *Ayuda* | *dinahosting* [en línea], 2019. [Consulta: 9 junio 2022]. Disponible en: https://dinahosting.com/ayuda/que-es-apache-y-para-que-sirve/.

ROBLEDANO, A., 2019. Qué es MySQL: Características y ventajas. *OpenWebinars.net* [en línea]. [Consulta: 9 junio 2022]. Disponible en: https://openwebinars.net/blog/que-es-mysql/.

RODRÍGUEZ SÁNCHEZ, T., 2015. *Metodología de desarrollo para la Actividad productiva de la UCI.Universidad de las Ciencias Informáticas.* La Habana: Universidad de las Ciencias Informáticas.

ROSADO, E., 2019. ¿Qué son los《Servicios SEO》?→ Claves para elegir Agencia. *NeoAttack* [en línea]. [Consulta: 27 junio 2022]. Disponible en: https://neoattack.com/blog/servicios-seo/.

SCHMULLER, J., 1999. *Aprendiendo UML en 24 horas*. S.l.: s.n.

Search engine marketing statistics 2022. *Smart Insights* [en línea], 2022. [Consulta: 28 junio 2022]. Disponible en: https://www.smartinsights.com/search-engine-marketing/search-engine-statistics/.

Servidor web - IBM Docs. [en línea], 2022. [Consulta: 9 junio 2022]. Disponible en: https://prod.ibmdocs-production-dal-6099123ce774e592a519d7c33db8265e-0000.us-south.containers.appdomain.cloud/docs/es/was/9.0.5?topic=servers-introduction-web.

SHARMA, S., 2021. *Modern API Development with Spring and Spring Boot*. 35 Livery Street Birmingham B3 2PB, UK.: Packt Publishing Ltd. ISBN 978-1-80056-247-9.

Slim Framework. *Slim Framework* [en línea], 2022. [Consulta: 7 junio 2022]. Disponible en: https://www.slimframework.com/.

SOMMERVILLE, I., 2011. *Software engineering*. Ninth Edition. S.l.: s.n. ISBN 978-0-13-703515-1.

SUÁREZ, P. y FONTELA, C., 2003. *DOCUMENTACIÓN Y PRUEBAS EN EL DESARROLLO TRADICIONAL DEL SOFTWARE*. S.l.: s.n.

SUNARDI, A. y SUHARJITO, 2019. MVC Architecture: A Comparative Study Between Laravel Framework and Slim Framework in Freelancer Project Monitoring System Web Based. *4th International Conference on Computer Science and Computational Intelligence*. S.l.: s.n., DOI 10.1016/j.procs.2019.08.150.

Swagger Codegen. [en línea], 2022. [Consulta: 9 junio 2022]. Disponible en: https://swagger.io/tools/swagger-codegen/.

Swagger Editor. [en línea], 2022. [Consulta: 9 junio 2022]. Disponible en: https://swagger.io/tools/swagger-editor/.

Swagger UI. [en línea], 2022. [Consulta: 9 junio 2022]. Disponible en: https://swagger.io/tools/swagger-ui/.

Visual Paradigm. [en línea], 2022. [Consulta: 7 junio 2022]. Disponible en: https://www.visual-paradigm.com/.

Visual Studio Code. [en línea], 2022. [Consulta: 9 junio 2022]. Disponible en: https://code.visualstudio.com/.

What is Git: Features, Command and Workflow in Git [Updated]. *Simplilearn.com* [en línea], 2022. [Consulta: 10 junio 2022]. Disponible en: https://www.simplilearn.com/tutorials/git-tutorial/what-is-git.

What is GitLab and How to Use It? [2022 Edition] | Simplilearn. [en línea], 2022. [Consulta: 10 junio 2022]. Disponible en: https://www.simplilearn.com/tutorials/git-tutorial/what-is-gitlab.

YouTube. *ProgrammableWeb* [en línea], 2022. [Consulta: 9 junio 2022]. Disponible en: https://www.programmableweb.com/api/youtube.

Anexos

<Contenido de los anexos con igual tipo de fuente Arial, pero a tamaño 11 puntos e interlineado 1.0 puntos. Debe tratar de sólo utilizarse aquellos anexos imprescindibles para complementar lo presentado en la memoria escrita y que no excedan las ocho (8) o diez (10 páginas). Deben aparecer uno a continuación del otro sin necesidad de saltos de página entre estos>

Anexo 1: Guía de observación al proceso de suscripción
Observador: Luis Enrique Reyes Pérez
Lugar: Dirección de proyectos especiales
Objetivo: conocer cómo se llevan a cabo los principales procesos dentro del proyecto, principalmente la suscripción, gestión de paquetes, vías de interacción con usuarios y proceso de evaluación de un sitio web.

1. ¿Qué procesos se llevan a cabo mediante SeoWebMas?
2. ¿Qué datos se recopilan y como se gestionan durante el proceso de suscripción?
3. ¿Cómo se calcula el precio de una suscripción?
4. ¿En qué consiste un paquete?
5. ¿Qué vías se utilizan para la comunicación con los usuarios del sistema?
6. ¿Cómo se lleva a cabo el proceso de evaluación de un sitio web?

Anexo 2: Entrevista realizada a especialistas del proyecto Monitor de Sitios Web cubanos
Objetivo: obtener información sobre los principales datos de SeoWebMas a integrar en los servicios ofrecidos por la propuesta de solución.

1. ¿Qué datos considera usted imprescindibles en SeoWebMas?
2. ¿Qué otro dato no ofrecido en SeoWebMas considera que debería estar disponible para su consumo?
3. ¿Cree usted que SeoWebMas integra correctamente los servicios del proyecto?

Anexo 3: Guía de estudio de la documentación del proceso de suscripción
Objetivo: comprender como se realiza el proceso de suscrición, que datos se necesitan obtener y de qué forma se procesan.

1. ¿Qué datos son recolectados durante el proceso de suscripción?
2. ¿Hay algún dato no obligatorio?
3. ¿Cuál es la fórmula del cálculo del precio de la suscripción?

¿Cómo se renueva una

Anexo 3: Historias de usuario

Historia de Usuario		
Número: 1	**Nombre de la historia:** Mostrar listado de sitios web	
Programador: Luis Enrique Reyes Pérez		**Iteración Asignada:** 1
Prioridad en negocio: Alta		**Tiempo estimado:** 1 semana

Riesgo en desarrollo: No aplica	**Tiempo real:** 1 semana
Descripción: Permite mostrar un listado con los sitios web que el usuario tiene con una suscripción aprobada, los sitios que le han sido compartido y los sitios en los que cuenta con permisos especiales. Todo esto se hace consultando 3 tablas en la base de datos, devuelve los listados separados.	
Observaciones: • El usuario debe estar autenticado • Para acceder a la URL debe utilizar el método GET.	

Historia de Usuario	
Número: 2	**Nombre de la historia:** Mostrar evaluación de sitio web.
Programador: Luis Enrique Reyes Pérez	**Iteración Asignada:** 1
Prioridad en negocio: Alta	**Tiempo estimado:** 2 semana
Riesgo en desarrollo: No aplica	**Tiempo real:** 2 semana
Descripción: Permite mostrar la evaluación de un sitio web desglosada por variables, cada una evaluada, medido su impacto, complejidad, identificado el grupo al que pertenece y recomendaciones para mejorar. Para lograrlo consulta 2 servicios, en el primero obtiene los grupos de variables y en el segundo la evaluación del sitio, estos resultados se parsean y se le añade el veredicto de la variable en correspondencia con la tabla *judgment* de la base de datos.	
Observaciones: • El usuario debe estar autenticado • Para acceder a la URL debe utilizar el método GET.	

Historia de Usuario	
Número: 3	**Nombre de la historia:** Mostrar notificaciones públicas
Programador: Luis Enrique Reyes Pérez	**Iteración Asignada:** 1
Prioridad en negocio: Media	**Tiempo estimado:** 1 semana
Riesgo en desarrollo: No aplica	**Tiempo real:** 1 semana
Descripción: Permite mostrar las notificaciones públicas del sistema y que aún no estén vencidas. Para hacerlo se consulta la base de datos y se extraen las notificaciones que no estén vencidas(*show before* > fecha actual)	
Observaciones: • Para acceder a la URL debe utilizar el método GET.	

Historia de Usuario

Número: 4	**Nombre de la historia:** Mostrar notificaciones privadas
Programador: Luis Enrique Reyes Pérez	**Iteración Asignada:** 1
Prioridad en negocio: Media	**Tiempo estimado:** 1 semana
Riesgo en desarrollo: No aplica	**Tiempo real:** 1 semana
Descripción: Permite mostrar las notificaciones de un usuario en particular y que aún no estén vencidas. Para hacerlo se consulta la base de datos y se extraen las notificaciones que no estén vencidas (*show before* > fecha actual) cuyo *inbox_notification.inbox_id* sea igual al email del usuario autenticado.	
Observaciones: • El usuario debe estar autenticado • Para acceder a la URL debe utilizar el método GET.	

Historia de Usuario	
Número: 8	**Nombre de la historia:** Editar suscripción
Programador: Luis Enrique Reyes Pérez	**Iteración Asignada:** 1
Prioridad en negocio: Alta	**Tiempo estimado:** 2 semana
Riesgo en desarrollo: No aplica	**Tiempo real:** 2 semana
Descripción: Permite al usuario editar los datos de una suscripción. Se recibe un listado con los sitios a añadir a la suscripción, el paquete contratado, así como una serie de datos necesarios para procesar la suscripción. Se debe verificar que los sitios recibidos estén en el directorio de no estarlo se le notificara al usuario mediante correo y a los encargados del centro para su procesamiento y posterior adición al mismo. Si todos los datos están correctos se hace un cálculo del precio, se actualizan los datos de la suscripción y notifica al usuario del éxito del proceso mediante correo electrónico.	
Observaciones: • El usuario debe estar autenticado • Para acceder a la URL debe utilizar el método PUT.	

Historia de Usuario	
Número: 9	**Nombre de la historia:** Renovar suscripción
Programador: Luis Enrique Reyes Pérez	**Iteración Asignada:** 1
Prioridad en negocio: Alta	**Tiempo estimado:** 2 semana
Riesgo en desarrollo: No aplica	**Tiempo real:** 2 semana

Descripción: Permite al usuario renovar una suscripción que haya terminado su tiempo de vigencia. Establece el estado de la suscripción especificada en Negociación, a la espera de pago para ser Aprobada nuevamente y se alargan las fechas de inicio y fin.
Observaciones: • El usuario debe estar autenticado • Para acceder a la URL debe utilizar el método PATCH.

Historia de Usuario	
Número: 10	**Nombre de la historia:** Obtener suscripción
Programador: Luis Enrique Reyes Pérez	**Iteración Asignada:** 1
Prioridad en negocio: Media	**Tiempo estimado:** 1 semana
Riesgo en desarrollo: No aplica	**Tiempo real:** 1 semana
Descripción: Permite que el usuario obtenga los datos de una suscripción especificada.	
Observaciones: • El usuario debe estar autenticado • Para acceder a la URL debe utilizar el método GET.	

Historia de Usuario	
Número: 11	**Nombre de la historia:** Compartir sitio web
Programador: Luis Enrique Reyes Pérez	**Iteración Asignada:** 1
Prioridad en negocio: Alta	**Tiempo estimado:** 2 semana
Riesgo en desarrollo: No aplica	**Tiempo real:** 2 semana
Descripción: Permite que el usuario comparta un sitio web de una de sus subscripciones con otro usuario especificado mediante un correo. Se debe verificar que el usuario tiene una suscripción con el sitio en específico, se agrega la entrada a la base de datos en la tabla *website_property_invitation_new* y notifica al usuario que recibe la invitación.	
Observaciones: • El usuario debe estar autenticado • Para acceder a la URL debe utilizar el método POST.	

Historia de Usuario	
Número: 12	**Nombre de la historia:** Autenticar usuarios
Programador: Luis Enrique Reyes Pérez	**Iteración Asignada:** 1
Prioridad en negocio: Alta	**Tiempo estimado:** 1 semana

Riesgo en desarrollo: No aplica	**Tiempo real:** 1 semana
Descripción: Permite autenticar a los usuarios que consuman del servicio. La autenticación se realizará mediante un middleware que use JWT como método para identificar el usuario.	
Observaciones: • El usuario debe estar autenticado • Para acceder a la URL debe utilizar el método POST.	

Historia de Usuario		
Número: 13	**Nombre de la historia:** Mostrar código de Telus	
Programador: Luis Enrique Reyes Pérez		**Iteración Asignada:** 1
Prioridad en negocio: Alta		**Tiempo estimado:** 1 semana
Riesgo en desarrollo: No aplica		**Tiempo real:** 1 semana
Descripción: Permite mostrar el código de telus de un sitio web. Verifica si el sitio tiene *host_telus* establecido en la base de datos, de no ser así llama al servicio de Telus encargado de añadir el sitio a telus para luego pedir el código.		
Observaciones: • El usuario debe estar autenticado • Para acceder a la URL debe utilizar el método GET.		