UNIVERSIDAD NACIONAL DEL ALTIPLANO - PUNO

FACULTAD DE INGENIERÍA ESTADÍSTICA E INFORMÁTICA

ESCUELA PROFESIONAL DE INGENIERÍA ESTADÍSTICA E INFORMÁTICA

TESIS

INTERCONECTIVIDAD BASADO EN API REST EN APLICACIONES DE LA MUNICIPALIDAD PROVINCIAL DE LAMPA

PRESENTADO POR:

BACH. ATENCIO FLORES DILMERD
BACH. MAMANI MACHACA DAVID

PARA OPTAR EL TÍTULO PROFESIONAL:

INGENIERO ESTADÍSTICO E INFORMÁTICA

PUNO – PERÚ

2017

UNIVERSIDAD NACIONAL DEL ALTIPLANO - PUNO
FACULTAD DE INGENIERÍA ESTADÍSTICA E INFORMÁTICA
ESCUELA PROFESIONAL DE INGENIERÍA ESTADÍSTICA E INFORMÁTICA

Interconectividad basado en API REST en aplicaciones web de la municipalidad provincial de Lampa

TESIS

PRESENTADA POR:

ATENCIO FLORES DILMERD
MAMANI MACHACA DAVID

PARA OPTAR EL TÍTULO PROFESIONAL DE:

INGENIERO ESTADÍSTICO E INFORMÁTICO

APROBADA POR:

PRESIDENTE : _____
Dr. Bernabe Canqui Flores

PRIMER MIEMBRO : _____
M.Sc. Charles Ignacio Mendoza Mollocondo

SEGUNDO MIEMBRO : _____
Dr. Leonel Coyla Idme

DIRECTOR / ASESOR : _____
M.Sc. Ernesto Nayer Tumi Figueroa

ÁREA : Base de datos y sistemas de información
TEMA : Ciencias de la información
FECHA: 2017/08/25

DEDICATORIA

Dedico este proyecto de tesis a mi madre y familia personas más importantes de mi vida que constantemente me ayudaron a realizar mis proyectos

A mis compañeros de trabajo con los cuales compartimos conocimientos y experiencias, lo que constituye un aliento y ánimo para la realización del presente trabajo

Dilmerd

La presente tesis está dedicada a mis padres, a mis hermanas y familiares, porque ellos siempre estuvieron ahí a mi lado brindándome su apoyo y sus consejos para hacer de mí una mejor persona.

A mis compañeros y amigos presentes y pasados, quienes compartieron sus conocimientos, alegrías y tristezas y a todas aquellas personas que estuvieron a mi lado apoyándome.

David

AGRADECIMIENTOS

A la Facultad de Ingeniería Estadística e Informática de la Universidad Nacional del Altiplano - Puno, por cobijarnos en sus aulas estos cinco años de formación.

A los Catedráticos de la Escuela Profesional de Ingeniería Estadística e Informática, por compartir sus conocimientos con sus estudiantes y contribuir en la formación profesional, por absolver cada uno de mis dudas, por su paciencia y calma en las sesiones de aprendizaje, mi cariño, respeto y admiración por cada uno de ellos.

Un agradecimiento muy grande también a los jurados M.Sc. Charles Ignacio Mendoza Mollocondo, Dr. Bernabe Canqui Flores y al M.Sc Leonel Coyla Idme y a nuestro asesor de Tesis M.Sc Ernesto Nayer Tumi Figueroa que participaron en mi gran formación profesional.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ÍNDICE DE ACRÓNIMOS

JSON : JavaScript Object Notation, objetos ligeros para el intercambio de información

API : Aplication Program Interface o de sus siglas en inglés: Interfaz programable de aplicaciones.

RPC : Remote Procedure Call o de sus siglas en inglés llamada de funciones o procedimientos remotos.

RESUMEN

El presente trabajo intitulado “IMPLEMENTACIÓN BASADO EN API REST PARA LA MEJORA DE LA INTERCONECTIVIDAD EN LAS APLICACIONES WEB DE LA MUNICIPALIDAD PROVINCIAL DE LAMPA”, tiene como objetivo el desarrollar una interfaz de comunicación entre aplicaciones de la municipalidad las que además de ser pocas aún conservan el carácter documentario clásico, por lo que para los futuras proyectos de desarrollo y el surgimiento de la interfaz de programación de aplicaciones (API). Para los líderes empresariales, las API representan la oportunidad de abrir nuevos flujos de ingresos y maximizar el valor del cliente. Pero los arquitectos empresariales son los que están a cargo de crear las API que hacen que los sistemas administrativos estén disponibles para volver a utilizarlos en nuevas aplicaciones web y móviles. Para el modelamiento de los diversos elementos de comunicación y programación para el WEB SERVICE basado en API REST, se ha hecho uso del lenguaje de Modelamiento Unificado UML y como metodología SCRUM para los documentar los procesos de desarrollo, así como la validación del mismo se procede con documentar la norma de validación de software ISO 9126. Como conclusión una vez finalizado el proyecto se logró implementar y desarrollar un prototipo de comunicación basado en API REST en la mejora de las aplicaciones de la Municipalidad Provincial de Lampa mediante entrevistas al personal que labora en las diferentes unidades administrativas evidenciando la mejora sustancial en los tiempos y procesos.

Palabras clave: API REST, Interconectividad, Aplicaciones, Municipalidad y calidad de servicio

ABSTRACT

The present work entitled "IMPLEMENTATION BASED ON API REST FOR THE IMPROVEMENT OF THE INTERCONECTIVITY IN THE WEB APPLICATIONS OF THE PROVINCIAL MUNICIPALITY OF LAMPA", aims to develop a communication interface between applications of the municipality which besides being few still conserve the classic documentary character, so for future development projects and the emergence of the application programming interface (API). For business leaders, APIs represent the opportunity to open new revenue streams and maximize customer value. But business architects are the ones in charge of creating the APIs that make administrative systems available for reuse in new web and mobile applications. For the modeling of the various elements of communication and programming for the WEB SERVICE based on REST API, the UML Unified Modeling language has been used and as a SCRUM methodology to document the development processes, as well as the validation of the same. With documenting the software validation standard ISO 9126

As conclusion after the project was completed, a communication model based on API REST was implemented and improved in the applications of the provincial municipality of Lampa through interviews with the staff working in the different administrative units evidencing the substantial improvement in the times and processes.

Keywords: API REST, Interconnectivity, Applications, Municipality, quality of service.

INTRODUCCIÓN

El surgimiento de la interfaz de programación de aplicaciones (API), representa una oportunidad única y un reto técnico. Para los municipios, las API representan la oportunidad de abrir nuevos flujos de información y maximizar el trabajo grupal en valor del cliente. La presente investigación se basa en crear las API que hacen que los sistemas administrativos estén disponibles para volver a utilizarlos en nuevas aplicaciones web y móviles.

Es fundamental que todas las partes interesadas comprendan que los objetivos administrativos y los retos técnicos de un programa de API están íntimamente relacionados. Los administradores del programa deben hacerse responsables de comunicar con claridad los objetivos clave de una API propuesta a los arquitectos que serán los que desarrollarán la interfaz.

Mientras tanto, los arquitectos deben asumir la responsabilidad de mantener un enfoque claro sobre estos objetivos a lo largo del proceso de implementación de una infraestructura de API y de diseño de la propia interfaz. Todas las decisiones técnicas deben contribuir a la creación de una interfaz que les permita a los desarrolladores crear aplicaciones que los usuarios finales realmente valorarán.

Es fundamental señalar que si lo que estás es ofreciendo un servicio online o estás realizando una aplicación web de gestión, entonces encaja perfectamente la filosofía de REST.

En el *CAPITULO I* se podrá encontrar la descripción general, así como el planteamiento del problema, la justificación, los objetivos, la hipótesis de la investigación y las limitaciones de la investigación

El *CAPITULO II*, se redactan los antecedentes de la investigación y a su vez se describe el marco teórico desarrollándose el estudio de investigación, se detalla la definición de términos básicos en el trabajo de investigación.

En el *CAPITULO III* se describe los materiales y métodos que se utilizaron para la realización del proyecto de investigación, describiendo las diferentes fases para la elaboración del proyecto.

En el *CAPITULO IV* de Resultados; se detalla los requerimientos del sistema y a su vez el diseño del mismo siempre buscando la mayor eficiencia del mismo y máxima comodidad y accesibilidad para el adecuado uso de los usuarios; también se realiza el seguimiento y validación de la información con prueba Estadística y las correspondientes recomendaciones para poder realizar trabajos a futuro.

CAPITULO I

PLAN DE INVESTIGACIÓN

1.1. DESCRIPCIÓN GENERAL

Actualmente vivimos en una época en que la tecnología web ha avanzado a un paso veloz, por este motivo vemos por conveniente que la reducción de tiempo en los procesos administrativos es fundamental para que estos se puedan realizar en menor tiempo y con más eficiencia, sin tener que estar buscando en todos los registros que con el paso del tiempo se fueron acumulando en archivos separados en la Municipalidad Provincial de LAMPA, redundando información de las personas que no solo laboran en la municipalidad sino también de los usuarios que constantemente hacen uso de los diversos servicios administrativos de la municipalidad, por este motivo se realizó el presente proyecto para que toda la información de los usuarios quede grabada en una base de datos evitando la duplicidad.

Para el desarrollo del presente proyecto, en primera instancia se planteó, Analizar los procesos internos de la parte administrativa del municipio para

poder conocer cómo funcionan los diversos procesos y hacia dónde van los datos que recolecta cada unidad administrativa. En esta etapa se utilizó el desarrollo de software ágil SCRUM, Se comienza con la visión general del producto, especificando y dando detalle a las funcionalidades o partes que tienen mayor prioridad de desarrollo y que pueden llevarse a cabo en un periodo de tiempo. Los cuales nos permitieron conceptuar la forma en que los datos fluyen, los procesos o transformaciones que sufren los datos y salida o resultado final del proceso.

Para después diseñar un Portal Web, En el proceso de diseño del prototipo de portal, lo primero que se hizo fue una investigación donde entre otros aspectos se definió la misión y visión de nuestro sitio, definición de las audiencias, las necesidades funcionales, posibles esquemas de organización, el contenido y la estructura del sitio. El cual nos llevó a obtener un Interfaz amigable.

Finalmente se propuso, Evaluar el Prototipo con los usuarios que interactúan con el Portal, en donde se hizo la prueba de aceptación introduciendo para ello datos reales para la prueba de Contenido

En cuanto a la implementación del prototipo del Portal Web se realizó en el lenguaje de programación PHP, usando Json. Teniendo un repositorio donde se guardan los datos que pedimos usando el API REST de Json

1.2. PLANTEAMIENTO DEL PROBLEMA

REST ha ganado una amplia aceptación a través de la Web como una alternativa más simple. Una evidencia clave de este cambio en diseño de

interfaces es la adopción de REST por las principales corriente de proveedores de servicios Web entre los cuales tenemos a Yahoo, Google, Facebook y Twitter, quienes han dejado de usar SOAP y WSDL a favor de un modelo más simple y orientado a recursos para exponer sus servicios.

Para el desarrollo de este proyecto, se expuso la información de prueba utilizando REST. Se diseñó la manera en que es entregada la información para ser consumida y la forma en que se puede tener acceso a esta información (URLs).

Para el lanzamiento de la aplicación, las implementaciones de los servicios quedaron a cargo de la universidad. Sin embargo, se respetó el diseño de las urls y estructura de la información.

El creciente aumento del uso de servicios web REST se debe a la simplicidad y ligereza que ofrece este estilo arquitectónico. Sin embargo, el proceso de composición es un proceso complejo que resulta difícil de automatizar por lo que mayoría de enfoques proponen métodos manuales o semiautomáticos.

Para poder aprovechar las características de la Web se diseñó *REST (Representational State Transfer)* que es un estilo arquitectónico especialmente diseñado para los sistemas distribuidos de hipermedios cuyo término acuñó Roy Thomas Fielding en su tesis doctoral. Actualmente, el número de servicios web basados en REST está creciendo y cada vez más empresas se decantan por este estilo arquitectónico debido a su mayor ligereza y simplicidad.

Esto nos conduce a la pregunta de investigación**: ¿En qué mejor manera el**

Modelo De Comunicación Basado En API REST Y Web Service en la Mejorará comunicación de Las Aplicaciones Web?

1.3. JUSTIFICACIÓN DE LA INVESTIGACIÓN

Actualmente la Municipalidad Provincial de Lampa tiene empleando 6 aplicaciones web para las oficinas de Recursos Humanos, Planeamiento, Contabilidad, Remuneraciones y Abastecimientos en los que se tiene ejecutándose aplicaciones desarrolladas en distintos tiempos y desarrolladores, usualmente por contrato por Locación de Servicios en el contrato para el desarrollo de las distintas aplicaciones, viste de un lado practico no tendría más trascendencia de no ser por el problema de la duplicidad de datos como nombres y apellidos de usuarios registrados, generando así una carga innecesariamente duplicada de 5000 registros esto multiplicado por la cantidad de aplicaciones generan un promedio de 30000 registros que podrían uniformizarse a través de una comunicación eficiente entre las aplicaciones.

La comunicación de aplicaciones por API REST basados en el estándar JSON permita una comunicación fluida y uniformizada sin errores de conversión de datos, soportando incluso la transferencia de datos BLOB (Binary Long Object) es decir formatos multimedia. Plantearemos en esta investigación un modelo de comunicación es decir un modelo de programación para la comunicación basada en una estructura de comunicación REST y JSON, presente en los lenguajes de procesamiento de datos Javascript, PHP y ASP.

La curva de aprendizaje del enfoque REST demuestra que en cortos períodos de tiempo es posible alcanzar importantes resultados, lo cual es

recomendable para personas que tienen poco conocimiento de las tecnologías web. Se obtuvo una aplicación que interactúa entre sí a través de simples solicitudes HTTP.

Toda la información provista por el Servicio Web puede ser recuperada desde diferentes aplicaciones, incluso ser utilizada como realimentación para otros sistemas.

Con el objetivo de realizar el menor impacto posible sobre las bases de datos de facultades, ambas aplicaciones (Servicio Web y Web móvil) fueron implementadas con ciertas limitaciones, sin embargo, se cumplieron exitosamente las expectativas.

1.4. OBJETIVOS

1.4.1. OBJETIVO GENERAL

- Desarrollar un Modelo de Comunicación Basado en API REST y Web Service en la mejora de las aplicaciones de la Municipalidad Provincial de Lampa.

1.4.2. OBJETIVOS ESPECÍFICOS

- Determinar la cantidad de usuarios y el número de trámites promedio que hacen uso de las aplicaciones web en cada oficina
- Establecer la tabla de protocolos de comunicación mediante nuestra API.

CAPITULO II

MARCO TEÓRICO

2.1. ANTECEDENTES DEL PROYECTO

LOCAL

Sistema Informático de Gestión Administrativa para la coordinación de Investigación de la Facultad de Ingeniería Estadística e Informática de la UNA PUNO 2014, Cuyo objetivo fue, la implementar un sistema para la gestión administrativa de la coordinación de investigación, de la FINESI. Y concluye que la implementación del sistema SIGACI permite realizar los procesos descritos en menor tiempo (Quispe, 2014).

NACIONAL

Ministerio de Economía y Finanzas del Perú en su portal datos abiertos explica porque es importante la verificación fehaciente de la información: Son datos producidos por la Entidades Públicas, puestas a disposición de la ciudadanía para su uso, re-uso y redistribución; siendo excluida la información protegida por la Ley de Transparencia y Acceso a la Información y demás normas

relacionadas a la materia. Los datos abiertos son accesibles de forma gratuita, a través de formatos reutilizables, que facilitan su análisis y permiten generar un nuevo producto o servicio innovador que contribuya al desarrollo económico del país. (http://datosabiertos.mef.gob.pe, 2017)

INTERNACIONAL

El creciente aumento del uso de servicios web REST se debe a la simplicidad y ligereza que ofrece este estilo arquitectónico. Este incremento propicia la necesidad de reutilizar los servicios existentes para componer servicios nuevos. Sin embargo, el proceso de composición es un proceso complejo que resulta difícil de automatizar por lo que mayoría de enfoques proponen métodos manuales o semiautomáticos.

El uso de servicios web basados en REST está aumentando y por tanto se hace cada vez más necesario el uso de mecanismos que permitan su composición. La composición de servicios web permite sintetizar un nuevo servicio mediante la reutilización de servicios existentes. Sin embargo, este proceso de composición resulta complejo de automatizar por lo que la mayoría de enfoques proponen soluciones manuales o semiautomáticas. Además, en muchos casos este proceso se realiza con servicios web basados en arquitecturas más complejas y pesadas como SOAP.

Teniendo en cuenta este escenario, el uso de servicios web basados en REST se presenta como una solución ideal para aprovechar las características que han hecho que la Web tenga tanto éxito y además permitir su composición.

Aunque existen múltiples enfoques para la composición de servicios web, no ocurre lo mismo cuando se trata de servicios web basados en el estilo arquitectónico REST.

Por tanto, la motivación de este trabajo de fin de máster surge de la necesidad de diseñare implementar una interfaz web que permita realizar la composición manual de servicios web REST y que también permita la gestión de las descripciones de estos servicios (Valero Menacho, 2015).

Debido al avance en tecnologías de información y telecomunicaciones, la tendencia a nivel mundial en sistemas de gobierno es migrar desde una democracia representativa a una democracia participativa y colaborativa. Esto es lo que se llama un Gobierno Abierto. Países como EEUU, Australia, Reino Unido, Finlandia y Suecia han mostrado avances significativos en esta materia.

Para que el Gobierno Abierto sea posible son necesarias varias cosas: voluntad y decisión política, apertura de datos públicos en formatos abiertos y estrategias de colaboración y participación ciudadana. Suponiendo que existe en Chile voluntad y decisión política, el paso siguiente es la apertura de datos públicos en formatos abiertos.

Una API es una interfaz de operación que permite la interacción entre diferentes partes de un sistema de software. En el caso específico de la API implementada para la SBIF, es un canal difusor de información pública en formatos abiertos y estándar, para el uso por parte de agentes externos.

La información disponible por medio de la API corresponde a información

pública que se encuentra disponible en el actual sitio web de la Superintendencia de Bancos e Instituciones Financieras. Esta información se compone de recursos como indicadores financieros, balances de instituciones financieras, etc.

Para la implementación de la API se usó la técnica de ingeniería de software REST, la cual hace uso sólo del protocolo HTTP para la entrega de recursos. Esto hace que operacionalmente la API sea eficiente en el uso de recursos y accesible a un amplio espectro de clientes. (González Orellana, 2010).

2.2. MARCO TEÓRICO

2.2.1. CIFRADO ELECTRÓNICO

Referenciando la información publicada por la Subsecretaría de Tecnologías de la Información (2013), Para que una persona natural o jurídica pueda firmar electrónicamente deberá generar un documento electrónico donde a través del uso de una firma electrónica, deberá extraer un algoritmo matemático denominado también función Hash, posteriormente a través de una clave privada la firma electrónica es encriptado generando así el documento firmado electrónicamente.

Tal documento contendrá toda la información del Hash generado el cual siempre será único debido a que se generó a través de una clave privada del titular, cada vez que el titular de la firma envía un mensaje, al llegar al destinatario empezará un proceso de descifrado del documento electrónico, lo cual se realizará a través de una des encriptación por medio de la clave pública del documento. Si el Hash es idéntico al del documento, entonces se

puede comprobar la validez del documento firmado electrónicamente. A continuación, se muestran algunos ejemplos del proceso del uso de una firma electrónica:

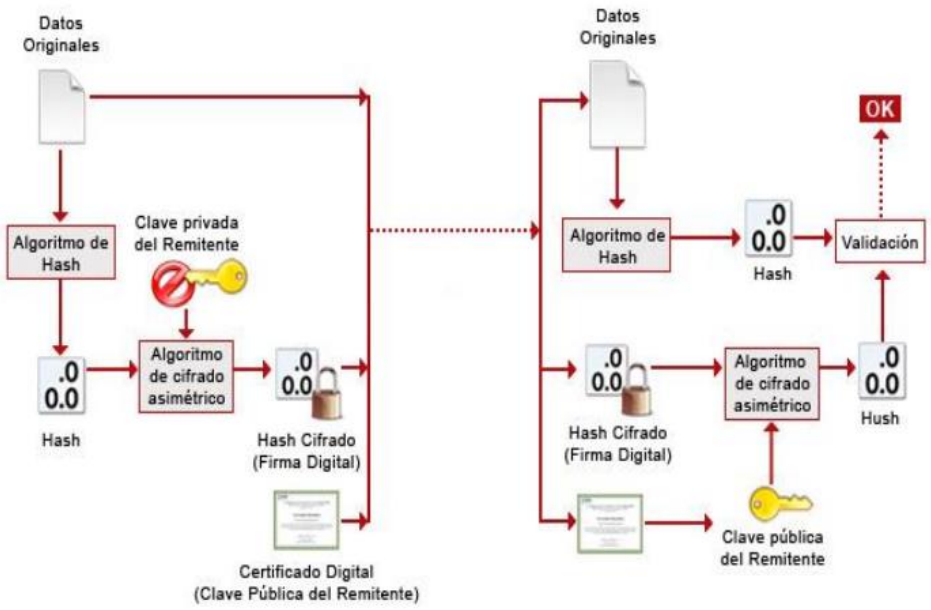

Figura 1: Como se firma electrónicamente

2.2.2. CODIGO QR

Un código QR (del inglés Quick Response code, "código de respuesta rápida") es la evolución del código de barras. Es un módulo para almacenar información en una matriz de puntos o en un código de barras bidimensional. La matriz se lee en el dispositivo móvil por un lector específico (lector de QR) y de forma inmediata nos lleva a una aplicación en internet y puede ser un mapa de localización, un correo electrónico, una página web o un perfil en una red social. Fue creado en 1994 por la compañía japonesa Denso Wave, subsidiaria de Toyota. Presenta tres cuadrados en las esquinas que permiten detectar la posición del código al lector. El objetivo de los creadores (un equipo de dos personas en Denso Wave, dirigido por Masahiro Hara) fue que el

código permitiera que su contenido se leyera a alta velocidad. Los códigos QR son muy comunes en Japón, donde los códigos bidimensionales son más populares.

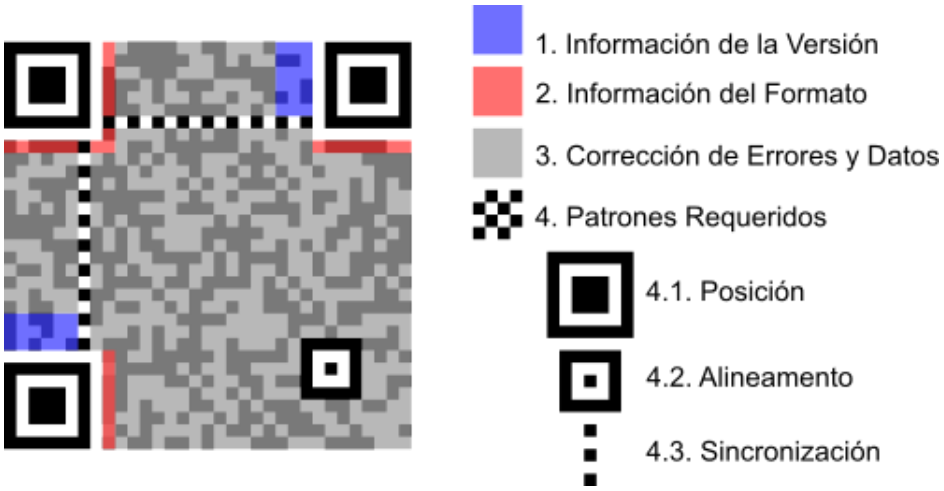

Figura 2: Taxonomía del código QR

2.2.3. WEB SERVICE / SERVICIO WEB

Un *web service* es un conjunto de protocolos y estándares que sirven para intercambiar datos entre aplicaciones. Distintas aplicaciones de software desarrolladas en lenguajes de programación diferentes, y ejecutadas sobre cualquier plataforma, pueden utilizar los servicios web para intercambiar datos en redes de ordenadores como internet.

De una manera más clara se podría decir que un web service es una función que diferentes servicios o equipos utilizan; es decir, solo se envían parámetros al servidor (lugar donde está alojado el web service) y éste responderá la petición. Entre algunas que se manejan de utilizar servicios webs en las aplicaciones destacan las siguientes:

Aportan interoperabilidad entre aplicaciones de software independientemente

de sus propiedades o de las plataformas sobre las que se instalen.

Los servicios Web fomentan los estándares y protocolos basados en texto, que hacen más fácil acceder a su contenido y entender su funcionamiento.

Al apoyarse en HTTP, los servicios Web pueden aprovecharse de los sistemas de seguridad firewall sin necesidad de cambiar las reglas de filtrado.

Permiten que servicios y software de diferentes compañías ubicadas en diferentes lugares geográficos puedan ser combinados fácilmente para proveer servicios integrados.

Permiten la interoperabilidad entre plataformas de distintos fabricantes por medio de protocolos estándar y abiertos. Las especificaciones son gestionadas por una organización abierta, la W3C, por tanto no hay secretismos por intereses particulares de fabricantes concretos y se garantiza la plena interoperabilidad entre aplicaciones.

La principal ventaja de utilizar un servicio web es que son bastante prácticos debido a que **son independientes de las aplicaciones**.

La mayoría de los sitios webs grandes (Facebook, MySpace, Microsoft) usan aplicaciones que utilizan servicios webs (web services).

2.2.4. API (Application Programming Interface)

La interfaz de programación de aplicaciones, abreviada como *API* del inglés: *Application Programming Interface*, es un conjunto de subrutinas, funciones y procedimientos (o métodos, en la programación orientada a objetos) que

ofrece cierta biblioteca para ser utilizado por otro *software* como una capa de abstracción.

Una API permite que un sitio Web brinde determinado servicio a otro, a través de llamadas a funciones documentadas y publicadas, facilitando de esta manera el "mash-up" o mezcla de servicios. Por ejemplo, hoy es posible que desde un blog personal se puedan publicar noticias del sitio de un famoso periódico, mezcladas con fotos que ya están alojadas en un sitio de fotografías, a través de llamadas a la API de estos dos servicios.

Una API detalla solamente la forma de llamar a cada función y la tarea que esta desempeña, sin importar cómo se lleva a cabo dicha tarea.

2.2.4.1. Detalles técnicos

Un API es un conjunto de reglas para escribir funciones o hacer llamados a subrutinas y acceder a otras funciones en una librería. Los programas que usan estas reglas o funciones en sus llamadas API pueden comunicarse con cualquiera que use dicha API.

Las API abren distintos tipos de diálogos con el proveedor para obtener o actualizar información en el mismo, entre ellos:

- Acceso a bases de datos
- Comunicación cliente/servidor
- Comunicación peer-to-peer
- Comunicación en tiempo real
- Event-driven (orientada a eventos)
- Store and forward

- Procesamiento de transacciones

Una API puede combinar recuperación de errores, traducción de datos, seguridad, manejo de colas y nomenclatura con una interface fácil de asimilar, que comprende acciones y comandos simples, pero con muchas opciones.

Para invocar una API, el programa debe llamar a una función tipo "send", especificando parámetros para el nombre de destino, indicadores de datos y opciones de confirmación.

Igual que una interfaz de usuario permite la interacción y comunicación entre un *software* y una persona, una API (acrónimo de *Application Programming Interface*) facilita la relación entre dos aplicaciones para **el intercambio de mensajes o datos**. Un conjunto de funciones y procedimientos que ofrece una biblioteca para que otro *software* la utilice como capa de abstracción, un espacio de acceso e intercambio de información adicional en la parte superior. Así una se sirve de la información de la otra sin dejar de ser independientes.

Cada API está diseñada en un lenguaje de programación concreto y con unas especificaciones distintas que la definen (las APIs pueden incluir especificaciones para **estructuras de datos y rutinas, clases de objetos o variables**, a partir de las cuales se basa el uso de esa interfaz). Además, suele ser habitual que cada una de ellas disponga de documentación completa y eficaz (un conjunto de tutoriales, manuales y reglas de buenas prácticas para esa interfaz de programación).

2.2.4.2. APIs de servicios web:

Son las interfaces de desarrollo de aplicaciones que permiten el intercambio de información entre un servicio web (*software* que da acceso a un servicio concreto a través de una URL) y una aplicación. Normalmente ese intercambio se produce a través de peticiones HTTP o HTTPS (la versión cifrada del protocolo HTTP). En la petición de la aplicación y respuesta, también en HTTP del servicio web, se contiene información de todo tipo tanto en los metadatos de la cabecera como en los del mensaje, **normalmente en dos tipos de formatos muy usados: XML o JSON**.(William, 2014)

2.2.5. API REST (Representational State Transfer)

La Transferencia de Estado Representacional (*en inglés Representational State Transfer*) o REST es un estilo de arquitectura software para sistemas hipermedia distribuidos como la World Wide Web. El término se originó en el año 2000, en una tesis doctoral sobre la web escrita por **Roy Fielding**, uno de los principales autores de la especificación del protocolo HTTP y ha pasado a ser ampliamente utilizado por la comunidad de desarrollo.

El consorcio W3C define los Servicios Web como sistemas software diseñados para soportar una interacción interoperable maquina a máquina sobre una red. Los Servicios Web suelen ser APIs Web que pueden ser accedidas dentro de una red (principalmente Internet) y son ejecutados en el sistema que los aloja.

La definición de Servicios Web propuesta alberga muchos tipos diferentes de sistemas, pero el caso común de uso de refiere a clientes y servidores que se

comunican mediante mensajes XML que siguen el estándar SOAP.

En los últimos años se ha popularizado un estilo de arquitectura Software conocido como REST (Representational State Transfer). Este nuevo estilo ha supuesto una nueva opción de estilo de uso de los Servicios Web. A continuación se listan los tres estilos de usos más comunes:

Remote Procedure Calls (RPC, Llamadas a Procedimientos Remotos): Los Servicios Web basados en RPC presentan una interfaz de llamada a procedimientos y funciones distribuidas, lo cual es familiar a muchos desarrolladores. Típicamente, la unidad básica de este tipo de servicios es la operación WSDL (WSDL es un descriptor del Servicio Web, es decir, el homólogo del IDL para COM).

Arquitectura Orientada a Servicios (Service-oriented Architecture, SOA). Los Servicios Web pueden también ser implementados siguiendo los conceptos de la arquitectura SOA, donde la unidad básica de comunicación es el mensaje, más que la operación. Esto es típicamente referenciado como servicios orientados a mensajes.

REST (Representation State Transfer). Los Servicios Web basados en REST intentan emular al protocolo HTTP o protocolos similares mediante la restricción de establecer la interfaz a un conjunto conocido de operaciones Rafael Navarro Marset. Modelado, Diseño e Implementación de Servicios Web 2006-07 REST vs Web Services 4/19 estándar (por ejemplo GET, PUT, etc.). Por tanto, este estilo se centra más en interactuar con recursos con estado, que con mensajes y operaciones. (Navarro M., Rafael, 2016)

REST (Representational State Transfer) es un estilo de arquitectura de software para sistemas hipermedias distribuidos tales como la Web. El término fue introducido en la tesis doctoral de Roy Fielding en 2000, quien es uno de los principales autores de la especificación de HTTP. *(Navarro M., Rafael, 2016)*.

En realidad, REST se refiere estrictamente a una colección de principios para el diseño de arquitecturas en red. Estos principios resumen como los recursos son definidos y diseccionados. El término frecuentemente es utilizado en el sentido de describir a cualquier interfaz que transmite datos específicos de un domino sobre HTTP sin una capa adicional, como hace SOAP. Estos dos significados pueden chocar o incluso solaparse. Es posible diseñar un sistema software de gran tamaño de acuerdo con la arquitectura propuesta por Fielding sin utilizar HTTP o sin interactuar con la Web. Así como también es posible diseñar una simple interfaz XML+HTTP que no sigue los principios REST, y en cambio seguir un modelo RPC. *(Navarro M., Rafael, 2016)*

Cabe destacar que REST no es un estándar, ya que es tan solo un estilo de arquitectura. Aunque REST no es un estándar, está basado en estándares:

HTTP

URL

Representación de los recursos: XML/HTML/GIF/JPEG/…

Tipos MIME: text/xml, text/html.

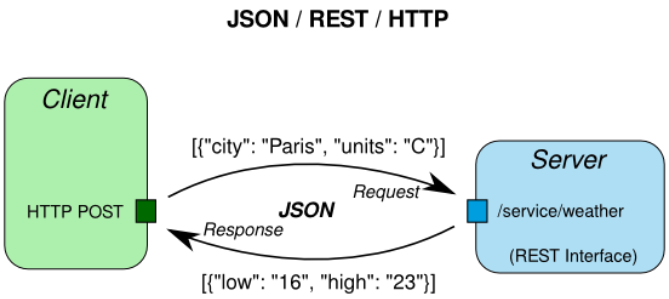

Figura 3: Esquema de Transacción de Datos API REST sobre el protocolo HTTP

2.2.6. ARQUITECTURA BASADA EN REST

De nuevo tomaremos como ejemplo a la Web. La Web evidentemente es un ejemplo clave de diseño basado en REST, ya que muchos principios son la base de REST. Posteriormente mostraremos un posible ejemplo real aplicado a Servicios Web.

La Web consiste del protocolo HTTP, de tipos de contenido, incluyendo HTML y otras tecnologías tales como el Domain Name System (DNS). Por otra parte, HTML puede incluir javascript y applets, los cuales dan soporte al codeon-demand, y además tiene implícitamente soporte a los vínculos. HTTP posee un interfaz uniforme para acceso a los recursos, el cual consiste de URIs, métodos, códigos de estado, cabeceras y un contenido guiado por tipos MIME.

Los métodos HTTP más importantes son PUT, GET, POST y DELETE. Ellos suelen ser comparados con las operaciones asociadas a la tecnología de base de datos, operaciones CRUD: CREATE, READ, UPDATE, DELETE. Otras

analogías pueden también ser hechas como con el concepto de copiar-y-pegar (Copy&Paste). Todas las analogías

Se representan en la siguiente tabla:

Acción	HTTP	SQL	Copy&Paste	Unix Shell
Create	PUT	Insert	Pegar	>
Read	GET	Select	Copiar	<
Update	POST	Update	Pegar después	>>
Delete	DELETE	Delete	cortar	Del/rm

Figura 4: Protocolo de Comunicación REST y equivalencia de formatos.

Las acciones (verbos) CRUD se diseñaron para operar con datos atómicos dentro del contexto de una transacción con la base de datos. REST se diseña alrededor de transferencias atómicas de un estado más complejo, tal que puede ser visto como la transferencia de un documento estructurado de una aplicación a otra. El protocolo HTTP separa las nociones de un servidor y un navegador. Esto permite a la implementación cada uno variar uno del otro, basándose en el concepto cliente/servidor. Cuando utilizamos REST, HTTP no tiene estado. Cada mensaje contiene toda la información necesaria para comprender la petición cuando se combina el estado en el recurso. Como resultado, ni el cliente ni el servidor necesita mantener ningún estado en la comunicación. Cualquier estado mantenido por el servidor debe ser modelado como un recurso. *(Pautasso et al., 2014)*

La restricción de no mantener el estado puede ser violada mediante cookies que mantienen las sesiones. Fielding advierte del riesgo a la privacidad y seguridad que frecuentemente surge del uso de cookies, así como la confusión y errores que pueden resultar de las interacciones entre cookies y el uso del botón “Go back” del navegador. HTTP proporciona mecanismos

para el control del caching y permite que ocurra una conversación entre el navegador y la caché del mismo modo que se hace entre el navegador y el servidor Web. *(Pautasso et al., 2014)*

2.2.6.1. ¿Qué tecnologías de servidor usas para implementar REST?

Es independiente, podrás tener un servidor que trabaja con PHP, Java, Python, NodeJS o lo que prefieras, o te imponga el proyecto. Tanto el lenguaje como la base de datos son importantes, porque nos sirven para procesar la solicitud y generar la respuesta, pero no importa cómo lo hagas en el servidor. Simplemente que la respuesta la entregues en ese lenguaje de intercambio de información que estés usando, generalmente JSON.

2.2.6.2. ¿Qué librerías JS?

Realmente tampoco estamos obligados a usar ninguna. Lo que estará claro es que si estás produciendo una web en el cliente usarás Javascript, pero la librería que tengas para facilitarte las cosas es indiferente.

Actualmente, por posibilidades, por comunidad y por facilidad de desarrollo los profesionales se están decantando por AngularJS, pero realmente usarás aquella con la que te sientas a gusto.

2.2.6.3. ¿Alguna librería del lado del servidor?

Debe haber cientos, también dependiendo del lenguaje que estés usando en el servidor. Si es con PHP puedes usar Laravel, Symfony o microframeworks como Slim. Si estás usando NodeJS probablemente encuentres útil Express

o Sails.js. Son solo por nombrar algunas, puesto que en este caso todo depende mucho de la tecnología que estés usando en el servidor.

2.2.6.4. ¿Qué tipo de aplicaciones son adecuadas para esta arquitectura?

Esta es una buena pregunta, porque realmente el desarrollo basado en API REST no es para cualquier tipo de proyecto. Si estás haciendo un sitio web y planeas que tu sitio va a ser siempre eso "una web", quizás sea innecesario desarrollar en base a un API. Por ejemplo, un sitio centrado en contenido, como un blog o una página de una empresa donde ofrece sus productos, servicios y modos de contacto, no tiene sentido desarrollarla en base a un API. Ahora, si lo que estás es ofreciendo un servicio online o estás realizando una aplicación web de gestión, entonces encaja perfectamente la filosofía de REST.

En general si piensas que tu sistema en el futuro podría ser accedido no solo desde una página web, sino también desde una App para móvil o desde una aplicación de otro tipo, las ventajas de REST serán especialmente útiles. En resumen, si sospechas que los datos o servicios que estás ofreciendo en un futuro puedan llegar a ser consultados desde otros sistemas, te interesa usar REST. Incluso hay profesionales que piensan que, aunque de momento no estés pensando en que esos datos o servicios puedan llegar a ser consultados desde otros sistemas ajenos a tu web, merece la pena usar REST porque es la solución que mayor escalabilidad te va a aportar. Todo esto lo verás mejor enseguida que tratemos las ventajas / inconvenientes. (Alvarez, 2014)

2.2.6.5. Ventajas del desarrollo basado en REST

Enumeramos una serie de ventajas que encontrarás en un desarrollo basado en API REST.

1. **separación cliente/servidor**

Al ser sistemas independientes (solo se comunican con un lenguaje de intercambio como JSON) puedes desarrollarlos proyectos autónomos, equipos autónomos. Al cliente le da igual cómo está hecha la API y al servidor le da igual qué vas a hacer con los datos que te ha proporcionado.

Si necesitas evolucionar/refactorizar uno de los dos, back o front, se puede hacer de manera separada, siempre que se mantenga la interfaz del API.

Puedes hacer varios front con un único backend (frontal no tiene xq ser solo web, puede ser un app para Android otro para un App iOS, etc.)

En este sentido otra ventaja importantísima es la posibilidad de crear tantos frontales como necesites con la misma API. Igual comienzas desarrollando una web, pero con la misma API podrás desarrollar también una aplicación para iOS, Android y si mañana gustas para Windows 8, Windows Phone, el próximo Windows 10, Blackberry, FirefoxOS y tantos como vengan en el futuro a encajar con tu estrategia de negocio. (Alvarez, 2014)

2. **Independencia de tecnologías / lenguajes**

Puedes desarrollar en cualquier tipo de tecnología o lenguaje con la que te sientas a gusto o con la que puedas acortar tus tiempos de desarrollo,

o encaje con la filosofía o necesidades de tu proyecto. Es indiferente que en el futuro cambies totalmente las tecnologías con las que está implementado tu API REST, siempre y cuando respetes "el contrato", o sea, que sigas teniendo las mismas operaciones en el API y hagan las mismas cosas que se supone que deben hacer. (Alvarez, 2014)

3. Fiabilidad, escalabilidad, flexibilidad

Al final solo te tienes que preocupar que el nexo cliente / servidor esté correcto. Puedes hacer cambios en tu servidor, lenguajes, bases de datos, etc. y mientras devuelvas los datos que toca todo irá correctamente.

Escalabilidad porque puedes crecer todo lo que necesites en cada momento. Tu API puede responder a otros tipos de operaciones o puede versionarse tanto como desees. También tu programación del lado del cliente puede crecer todo lo necesario con el tiempo, incluso como decíamos, crear otros frontales, no solo web, también de Apps para cualquier dispositivo.

A la hora de ejecutar tu aplicación también tienes una flexibilidad mucho mayor. Las páginas del front las puedes enviar desde unos servidores y las API pueden estar alojadas en servidores independientes, tantos como necesites. Por las características de REST (principalmente no guardar estado) es indiferente qué servidor atienda cada solicitud, pues es el propio cliente el que tiene que mandar el estado al servidor, así que el balanceo de carga es mucho más simple que en aplicaciones tradicionales donde el front está mezclado con el back. También tienes la aproximación de las "micro APIs", en las que puedes dividir los procesos

en diferentes servidores que atiendan a diferentes tipos de operaciones del API. (Alvarez, 2014)

4. Experiencia de usuario

Aunque eso depende más de cómo está hecha la parte del cliente, teóricamente el desarrollo de sitios web basados en un API puede dar mejor desempeño que uno tradicional. Cuando haces una solicitud al servidor lo que tienes como respuesta son datos planos, que requieren tiempos de transferencia menores que si esos mismos datos los recibieras mezclados con el HTML/CSS de la presentación. En este tipo de aplicaciones web no necesitas recargar la página, aunque esto no es una ventaja específica del desarrollo basado en REST, sino del uso de Ajax en general, con el que podemos conseguir aplicaciones web que se asemejan más a aplicaciones de escritorio. (Alvarez, 2014)

5. REST requiere menos recursos del servidor

Esto no es necesariamente cierto aunque en muchos casos sí se pueda deducir. Hay muchas opiniones alrededor de REST. Nosotros basamos esa afirmación en estos motivos:

No mantener el estado, no requiere memoria, se pueden atender más peticiones

No requiere escribir el HTML, por lo tanto tienes menos procesamiento en el servidor(Alvarez, 2014)

2.2.6.6. BUENAS PRÁCTICAS Y PATRONES DE DISEÑO BÁSICOS

Respeta la semántica de http

Lo más importante a la hora de diseñar una API REST es respetar la semántica de HTTP. Toda la infraestructura de internet está preparada para respetar dicha semántica, y la interoperabilidad de la web se basa en ella. Si nuestra API ignora este aspecto, no podrá inter operar con caches, navegadores, CDNs, robots (*crawlers*), y en general, con cualquier otro sistema que esté operando en la web.

Aunque esto parece obvio, existen muchos sistemas que no respetaban este principio. Durante mucho tiempo la mayoría de las aplicaciones web han usado POST para leer y navegar con los consiguientes problemas (*back button* del navegador roto, problemas de cache, y efectos inesperados) (Amodeo, 2013).

Servicios multimedia

Una ventaja de los servicios REST es que permite negociar el formato exacto en el que se va a intercambiar la información.

Esto nos abre un nuevo campo con el que hacer nuestros servicios más potentes. Usando la misma URI y los mismos métodos HTTP, podemos consumir el recurso en distintos formatos, no es necesario tener una URI diferente para cada formato (Amodeo, 2013).

Cache

La cache es uno de los mecanismos más importantes que tiene la web para asegurar la alta escalabilidad. Permite que el servidor no tenga que procesar

todas las peticiones, aumentando la escalabilidad de este. También permite a agentes intermedios, más cercanos (incluso en la misma máquina) al cliente, responder en vez del servidor. De esta manera la latencia de las peticiones baja considerablemente (Amodeo, 2013).

Multi idioma

Mediante el protocolo HTTP podemos construir un soporte multi idioma robusto en nuestros servicios REST.

Mediante la cabecera Accept-Charset el cliente puede especificar al servidor los juegos de caracteres qué soporta por orden de preferencia. Podemos especificar una lista de juegos de caracteres, e incluso especificar preferencia mediante el parámetro q. Se trata de un mecanismo similar al usado en la cabecera Accept para especificar preferencia en tipos MIME. Si no se define el juego de caracteres se escogerá ISO-8859-1 por defecto (Amodeo, 2013).

2.2.7. IMPLEMENTACIÓN DE APLICACIONES WEB

HTML5 está reemplazando previos complementos o plugins, como Flash o Java applets, por nuevas y mejores tecnologías ahora incluso WebGL y Canvas para representación gráfica de datos. Los desarrollos web contribuyen a la simplificación de la interfaz de desarrollo sobre el problema real de transportabilidad del programa final, como el caso particular de un correo electrónico o página web que únicamente requiere ejecutarse sobre un navegador web, dejando al desarrollador concentrarse en el algoritmo y el objetivo planteado. Las aplicaciones web están diseñadas para apoyar el desarrollo de sitios web dinámicos, aplicaciones y servicios web. Las

aplicaciones Web intentan aliviar el exceso de carga asociado con actividades comunes usadas en desarrollos web. Por ejemplo, muchos de ellos proporcionan bibliotecas para acceder a bases de datos, administración general de archivos y carpetas, mantenimiento de ellas, estructuras para plantillas y gestión de sesiones, y con frecuencia facilitan la reutilización de código. *(Pautasso et al., 2014)*

2.2.8. JSON (JavaScript Object Notation)

JSON, acrónimo de *JavaScript Object Notation*, es un formato de texto ligero para el intercambio de datos. JSON es un subconjunto de la notación literal de objetos de JavaScript aunque hoy, debido a su amplia adopción como alternativa a XML, se considera un formato de lenguaje independiente.

Una de las supuestas ventajas de JSON sobre XML como formato de intercambio de datos es que es mucho más sencillo escribir un analizador sintáctico (parser) de JSON. En JavaScript, un texto JSON se puede analizar fácilmente usando la función eval(), lo cual ha sido fundamental para que JSON haya sido aceptado por parte de la comunidad de desarrolladores AJAX, debido a la ubicuidad de JavaScript en casi cualquier navegador web.

En la práctica, los argumentos a favor de la facilidad de desarrollo de analizadores o de sus rendimientos son poco relevantes, debido a las cuestiones de seguridad que plantea el uso de eval() y el auge del procesamiento nativo de XML incorporado en los navegadores modernos. Por esa razón, JSON se emplea habitualmente en entornos donde el tamaño del flujo de datos entre cliente y servidor es de vital importancia (de aquí su uso por Yahoo, Google, etc, que atienden a millones de usuarios) cuando la fuente

de datos es explícitamente de fiar y donde no es importante el no disponer de procesamiento XSLT para manipular los datos en el cliente.

Si bien es frecuente ver JSON posicionado *contra* XML, también es frecuente el uso de JSON y XML en la misma aplicación.

Por ejemplo, una aplicación de cliente que integra datos de Google Maps con datos meteorológicos en SOAP hacen necesario soportar ambos formatos.

En diciembre de 2005 Yahoo! comenzó a dar soporte opcional de JSON en algunos de sus servicios web.

Ni Yahoo, ni Google emplean JSON, sino LJS Una de las cualidades intrínsecas de Javascript denominada LJS (Literal Javascript) facilita el flujo de datos e incluso de funciones, para la cual no requiere la función eval() si son datos los que se transfieren como en el caso de XML. Todo lo referente a transferencia de datos en todos sus tipos, incluyendo arrays, booleans, integers, etc. no requieren de la función eval() y es precisamente en eso en donde supera por mucho JavaScript al XML, si se utiliza el LJS y no la incorrecta definición de JSON.

2.2.8.1. Sintaxis

Los tipos de datos disponibles con JSON son:

Números: Se permiten números negativos y opcionalmente pueden contener parte fraccional separada por puntos. Ejemplo: 123.456

Cadenas: Representan secuencias de cero o más caracteres. Se ponen entre doble comilla y se permiten cadenas de escape. Ejemplo: "Hola"

Booleanos: Representan valores booleanos y pueden tener dos valores: true y false

null: Representan el valor nulo.

Array: Representa una lista ordenada de cero o más valores los cuales pueden ser de cualquier tipo. Los valores se separan por comas y el vector se mete entre corchetes. Ejemplo ["juan","pedro","jacinto"]

Objetos: Son colecciones no ordenadas de pares de la forma <nombre>:<valor> separados por comas y puestas entre llaves. El nombre tiene que ser una cadena y entre ellas. El valor puede ser de cualquier tipo. Ejemplo:

{"departamento":8,"nombredepto":"Ventas","director": " juan rodriguez","empleados":[{"nombre":"Pedro","apellido":"Fernandez"}, {"nombre":"Jacinto","apellido":"Benavente"}]}

2.2.8.2. Modelos de procesamiento

Al ser JSON un formato muy extendido para el intercambio de datos, se han desarrollado API para distintos lenguajes (por ejemplo ActionScript, C, C++, C#, ColdFusion, Common Lisp, Delphi, E, Eiffel, Java, JavaScript, ML, Objective-C, Objective CAML, Perl, PHP, Python, Rebol, Ruby, Lua y Visual FoxPro) que permiten analizar sintácticamente, generar, transformar y procesar este tipo de datos.

Los modelos de programación más utilizados para tratar con JSON en los distintos lenguajes son:

Modelo de objeto.- El JSON completo es almacenado en memoria en un formato de árbol. Este árbol es navegado, analizado y modificado con las API apropiadas. Como lo carga todo en memoria y luego lo procesa este modelo consume muchos recursos. Sin embargo, es muy flexible para manipular el contenido. Este modelo es permitido por ejemplo en java por la JSR 353 y por la biblioteca Jackson

Modelo de flujo: Los datos son leídos o escritos en bloques. Por ejemplo, cada vez que se lee un bloque el analizador genera eventos apropiados para indicar el tipo de bloque de que se trata. El cliente puede procesar el contenido escuchando los eventos apropiados. Además, es el cliente el que decide cómo se va leyendo el JSON permitiendo parar o saltar contenidos en mitad del proceso. El proceso de escritura tiene propiedades análogas. Por ejemplo, este modelo es permitido en java por la JSR 353.

Convirtiendo los objetos JSON en objetos del lenguaje. En java esto es realizado por ejemplo por las bibliotecas Jackson y Gson.

Uso de JSON

En teoría, es trivial analizar JSON en JavaScript usando la función JSON.parse() incorporada en el lenguaje. Por ejemplo:

miObjeto = JSON.parse(json_datos);

En la práctica, las consideraciones de seguridad por lo general recomiendan no usar *eval* sobre datos crudos y debería usarse un analizador JavaScript distinto para garantizar la seguridad. El analizador proporcionado por JSON.org usa eval() en su función de análisis, protegiéndola con una

expresión regular de forma que la función sólo ve expresiones seguras.

Un ejemplo de acceso a datos JSON usando XMLHttpRequest es:

```
var http_request = new XMLHttpRequest();
var url = "http://example.net/jsondata.php";
// Esta URL debería devolver datos JSON
 // Descarga los datos JSON del servidor.
http_request.onreadystatechange = handle_json;
http_request.open("GET", url, true);
http_request.send(null);
 function handle_json() {
  if (http_request.readyState == 4) {
    if (http_request.status == 200) {
      var json_data = http_request.responseText;
      var the_object = eval("(" + json_data + ")");
    } else {
      alert("Ocurrio un problema con la URL.");
    }
    http_request = null;
  }
}
```

Obsérvese que el uso de XMLHttpRequest en este ejemplo no es compatible con todos los navegadores, ya que existen variaciones sintácticas para Internet Explorer, Opera, Safari, y navegadores basados en Mozilla.

También es posible usar elementos <iframe> ocultos para solicitar los datos de manera asíncrona, o usar peticiones <form target="url_to_cgi_script" />.

Estos métodos eran los más habituales antes del advenimiento del uso generalizado de XMLHttpRequest.

Hay una biblioteca[3] para el framework .NET que exporta clases .NET con la sintaxis de JSON para la comunicación entre cliente y servidor, en ambos sentidos.

2.2.8.3. Comparación con XML y otros lenguajes de marcado

XML goza de mucho menor soporte y ofrece menos herramientas de desarrollo (tanto en el lado del cliente como en el lado del servidor). Hay muchos analizadores JSON en el lado del servidor, existiendo al menos un analizador para la mayoría de los entornos. En algunos lenguajes, como Java o PHP, hay diferentes implementaciones donde escoger. En JavaScript, el análisis es posible de manera nativa con la función JSON.parse(). Ambos formatos carecen de un mecanismo para representar grandes objetos binarios.

Con independencia de la comparación con XML, JSON puede ser muy compacto y eficiente si se usa de manera efectiva. Por ejemplo, la aplicación DHTML de búsqueda en «BarracudaDrive» (en inglés). Archivado desde el original el 1 de diciembre de 2015. Recibe los listados de directorio como JSON desde el servidor. Esta aplicación de búsqueda está permanentemente consultando al servidor por nuevos directorios, y es notablemente rápida, incluso sobre una conexión lenta.

Los entornos en el servidor normalmente requieren que se incorpore una función u objeto analizador de JSON. Algunos programadores, especialmente

los familiarizados con el lenguaje C, encuentran JSON más natural que XML, pero otros desarrolladores encuentran su escueta notación algo confusa, especialmente cuando se trata de datos fuertemente jerarquizados o anidados muy profundamente.

Hay más comparaciones entre JSON y XML en JSON.org

YAML es un súper conjunto de JSON que trata de superar algunas de las limitaciones de éste. Aunque es significativamente más complejo, aún puede considerarse como ligero. El lenguaje de programación Ruby utiliza YAML como el formato de serialización por defecto. Así pues, es posible manejar JSON con bastante sencillez.

2.2.9. AJAX (Asynchronous JavaScript And XML)

AJAX son las siglas de Asynchronous JavaScript And XML, (Javascript asíncrono y XML). No es en sí un lenguaje de programación, sino una nueva técnica que combina varios lenguajes de programación.

La ventaja de AJAX respecto a otros lenguajes de programación web es la asincronía. Esto consiste en que cuando queremos intercambiar datos con el servidor (por ejemplo, enviar o comprobar un formulario, consultar una base de datos, etc.), la página no se queda parada esperando la respuesta, sino que se pueden seguir ejecutando acciones mientras tanto.

Con ajax podemos crear páginas interactivas. En éstas solicitamos datos al servidor, los cuales podemos tener guardados en otras páginas o en bases de datos. El servidor devuelve los datos, los cuales se cargan en la misma página y en segundo plano. Lo de "segundo plano" significa que mientras

esperamos que se reciban los datos la página no se queda parada, y el usuario o la programación de la página pueden seguir haciendo otras cosas.

Para poder entender este manual debemos tener conocimientos de HTML, CSS, y Javascript. Es conveniente también tener nociones de XML y acceso al DOM. Para las últimas páginas necesitamos también tener conocimientos de PHP, ya que es el lenguaje que se emplea para procesar los datos en el servidor.

2.2.9.1. Componentes de AJAX

Ajax es una combinación de los siguientes lenguajes de programación y elementos:

HTML (o XHTML) y CSS: Base para el diseño de la página.

DOM y Javascript: Forma de acceder dinámicamente a las partes de la página.

Objeto XMLHttpRequest: Es el que permite la comunicación asíncrona (en segundo plano) con el servidor.

XML : Formato en el que están los datos que se solicitan al servidor; aunque otros formatos también pueden funcionar, como son HTML, texto plano(txt), json ... etc.

PHP : En este manual trataremos también cómo mandar datos al servidor. Este los recoge mediante PHP. Una vez enviados pueden guardarse en una base de datos o procesarlos para enviar alguna información.

2.2.9.2. Cómo funciona ajax

Usando sólo PHP u otros lenguajes de servidor, al hacer una petición, el servidor realiza una serie de tareas y después nos devuelve los datos. Mientras se realiza este proceso la página permanece en espera, es decir está parada. Esto puede que no tenga importancia si se manejan pocos datos y el servidor tiene potencia para responder rápidamente. Sin embargo, si se manejan muchos datos o hay muchas peticiones a la vez (páginas muy visitadas), el tiempo de respuesta puede ser más largo. Mientras se espera la respuesta la página permanece parada.

Con ajax al trabajar de forma asíncrona, permite que el usuario pueda seguir haciendo otras cosas o la página pueda mostrar otras cosas mientras se produce la respuesta.

Los siguientes gráficos muestran la forma de trabajar pidiendo datos al servidor de forma síncrona (sin ajax) o asíncrona (con ajax):

Forma clásica de trabajar al interactuar con el servidor:

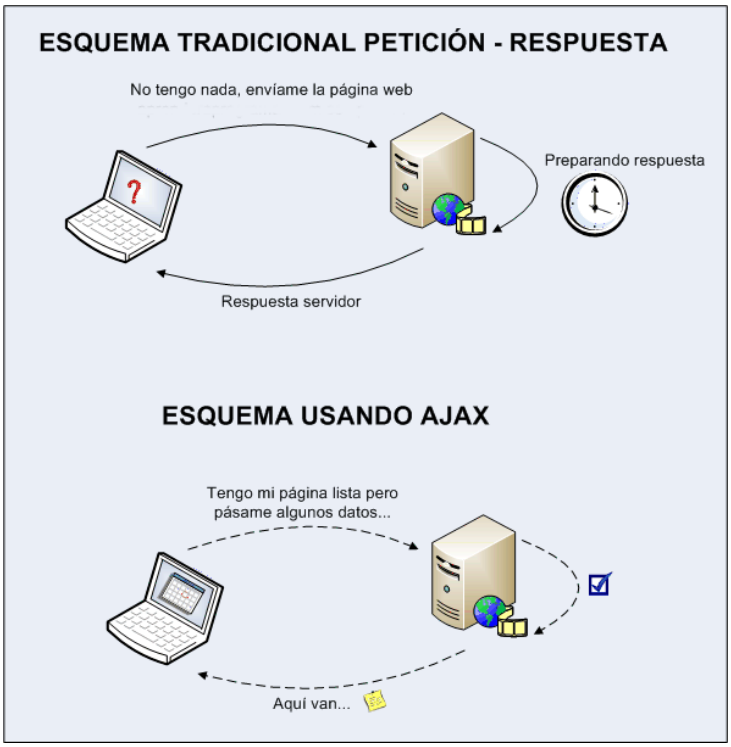

Figura 5: Forma de trabajar con AJAX interactuar con el servidor

Ajax es particularmente útil en páginas que manejan gran cantidad de datos o que son usadas por una gran cantidad de usuarios. Algunos ejemplos de páginas que usan ajax son Gmail (el correo de Google) o Google Maps.

CAPITULO III

MATERIALES Y MÉTODOS

3.1. METODOLOGÍA

El tipo de investigación que se desarrolló en el proyecto es del tipo cuasi-experimental, donde se identificara los efectos y resultados mostrados en la investigación, que se realizaron en la municipalidad provincial de Lampa. La metodología es factible dado que se pueden realizar en pequeñas unidades, por lo cual son más baratos y tienen menos obstáculos prácticos.

3.1.1. POBLACIÓN

Para la presente investigación la población estuvo constituida por un total de 18 trabajadores administrativos que trabajan en las 5 diferentes áreas y utilizan el aplicativo web en la municipalidad provincial de Lampa.

3.1.2. MÉTODO DE RECOPILACIÓN DE DATOS

La recopilación de datos para el presente trabajo de investigación se realizó a través de entrevistas a cada uno de los trabajadores de las diferentes áreas administrativas de la municipalidad provincial de Lampa.

3.1.3. MÉTODOS DE TRATAMIENTOS DE DATOS

El procedimiento para realizar las consultas y trámites en las diferentes áreas administrativas tiene una duración promedio de 3 a 4 días hábiles lo cual hace muy lenta la efectividad de mencionados trámites.

3.2. METODOLOGÍA DE DESARROLLO SCRUM

Scrum es un proceso en el que se aplican de manera regular un conjunto de buenas prácticas para trabajar colaborativamente, en equipo, y obtener el mejor resultado posible de un proyecto. Estas prácticas se apoyan unas a otras y su selección tiene origen en un estudio de la manera de trabajar de equipos altamente productivos.

En Scrum se realizan entregas parciales y regulares del producto final, priorizadas por el beneficio que aportan al receptor del proyecto. Por ello, Scrum está especialmente indicado para proyectos en entornos complejos, donde se necesita obtener resultados pronto, donde los requisitos son cambiantes o poco definidos, donde la innovación, la competitividad, la flexibilidad y la productividad son fundamentales.

Scrum también se utiliza para resolver situaciones en que no se está entregando al cliente lo que necesita, cuando las entregas se alargan demasiado, los costes se disparan o la calidad no es aceptable, cuando se necesita capacidad de reacción ante la competencia, cuando la moral de los equipos es baja y la rotación alta, cuando es necesario identificar y solucionar ineficiencias sistemáticamente o cuando se quiere trabajar utilizando un proceso especializado en el desarrollo de producto. Ver en detalle cuales son

los beneficios de Scrum, sus fundamentos y sus requisitos.

3.2.1. El proceso

En Scrum un proyecto se ejecuta en bloques temporales cortos y fijos (iteraciones que normalmente son de 2 semanas, aunque en algunos equipos son de 3 y hasta 4 semanas, límite máximo de feedback y reflexión). Cada iteración tiene que proporcionar un resultado completo, un incremento de producto final que sea susceptible de ser entregado con el mínimo esfuerzo al cliente cuando lo solicite.

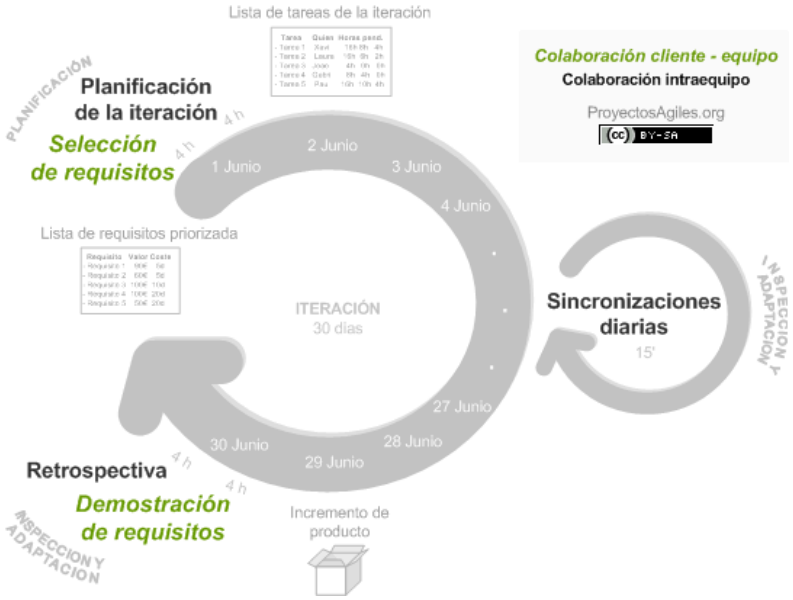

Figura 6: Metodología SCRUM iteraciones

Diagrama - proceso-SCRUM:

El proceso parte de la lista de objetivos/requisitos priorizada del producto, que actúa como plan del proyecto. En esta lista el cliente prioriza los objetivos balanceando el valor que le aportan respecto a su coste y quedan repartidos en iteraciones y entregas.

Las actividades que se llevan a cabo en Scrum son las siguientes:

Planificación de la iteración

El primer día de la iteración se realiza la reunión de planificación de la iteración. Tiene dos partes:

Selección de requisitos (4 horas máximo). El cliente presenta al equipo la lista de requisitos priorizada del producto o proyecto. El equipo pregunta al cliente las dudas que surgen y selecciona los requisitos más prioritarios que se compromete a completar en la iteración, de manera que puedan ser entregados si el cliente lo solicita.

Planificación de la iteración (4 horas máximo). El equipo elabora la lista de tareas de la iteración necesarias para desarrollar los requisitos a que se ha comprometido. La estimación de esfuerzo se hace de manera conjunta y los miembros del equipo se auto asignan las tareas.

Ejecución de la iteración

Cada día el equipo realiza una reunión de sincronización (15 minutos máximo). Cada miembro del equipo inspecciona el trabajo que el resto está realizando (dependencias entre tareas, progreso hacia el objetivo de la iteración, obstáculos que pueden impedir este objetivo) para poder hacer las adaptaciones necesarias que permitan cumplir con el compromiso adquirido. En la reunión cada miembro del equipo responde a tres preguntas:

¿Qué he hecho desde la última reunión de sincronización?

¿Qué voy a hacer a partir de este momento?

¿Qué impedimentos tengo o voy a tener?

Durante la iteración el Facilitador (SCRUM Master) se encarga de que el equipo pueda cumplir con su compromiso y de que no se merme su productividad.

Elimina los obstáculos que el equipo no puede resolver por sí mismo.

Protege al equipo de interrupciones externas que puedan afectar su compromiso o su productividad.

Durante la iteración, los clientes junto con el equipo refinan la lista de requisitos (para prepararlos para las siguientes iteraciones) y, si es necesario, cambian o re planifican los objetivos del proyecto para maximizar la utilidad de lo que se desarrolla y el retorno de inversión.

Inspección y adaptación

El último día de la iteración se realiza la reunión de revisión de la iteración. Tiene dos partes:

Demostración (4 horas máximo). El equipo presenta al cliente los requisitos completados en la iteración, en forma de incremento de producto preparado para ser entregado con el mínimo esfuerzo. En función de los resultados mostrados y de los cambios que haya habido en el contexto del proyecto, el cliente realiza las adaptaciones necesarias de manera objetiva, ya desde la primera iteración, re planificando el proyecto.

Retrospectiva (4 horas máximo). El equipo analiza cómo ha sido su manera de trabajar y cuáles son los problemas que podrían impedirle progresar

adecuadamente, mejorando de manera continua su productividad. El Facilitador se encargará de ir eliminando los obstáculos identificados.

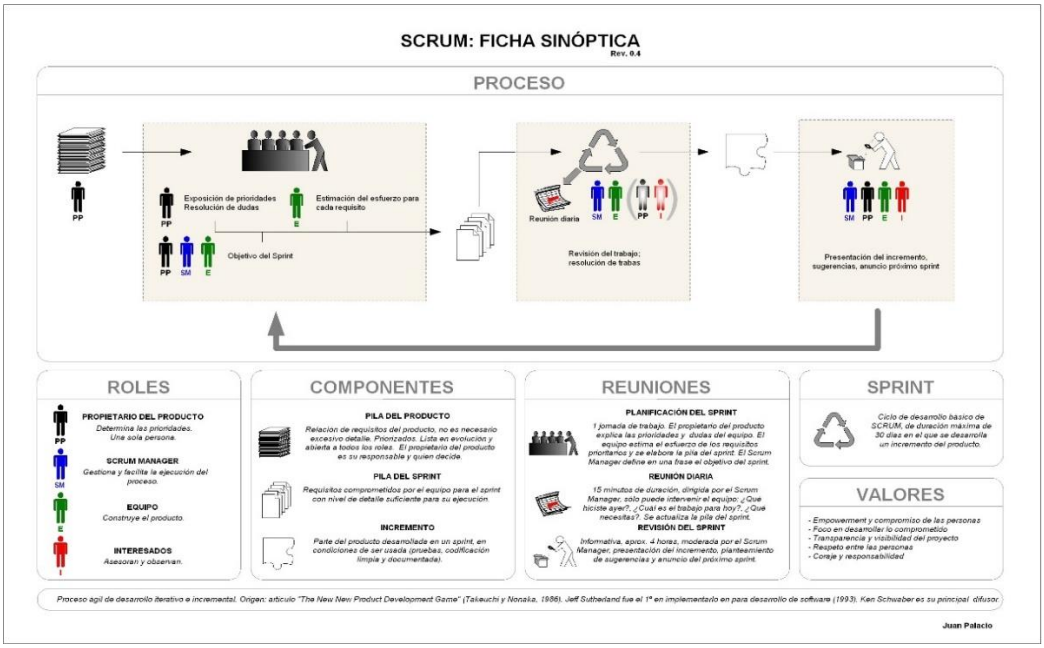

Figura 7 : SCRUM – Ficha Sinóptica

3.3. LENGUAJE DE MODELAMIENTO UNIFICADO – UML

El lenguaje unificado de modelado (UML, por sus siglas en inglés, Unified Modeling Language) es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad; está respaldado por el Object Management Group (OMG).

Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema. UML ofrece un estándar para describir un "plano" del sistema (modelo), incluyendo aspectos conceptuales tales como procesos, funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y compuestos reciclados.

Es importante remarcar que UML es un "lenguaje de modelado" para especificar o para describir métodos o procesos. Se utiliza para definir un

sistema, para detallar los artefactos en el sistema y para documentar y construir. En otras palabras, es el lenguaje en el que está descrito el modelo.

Se puede aplicar en el desarrollo de software gran variedad de formas para dar soporte a una metodología de desarrollo de software (tal como el Proceso Unificado Racional, Rational Unified Process o RUP), pero no especifica en sí mismo qué metodología o proceso usar.

UML, Ayuda a al usuario a entender la realidad de la tecnología y la posibilidad de que reflexione antes de invertir y gastar grandes cantidades en proyectos que no estén seguros en su desarrollo, reduciendo el coste y el tiempo empleado en la construcción de las piezas que constituirán el modelo (james Rumbaungh , Ivar Jacobson, 2000).

3.3.1. Bloques básicos de construcción de UML

Los bloques básicos de construcción de UML son tres, los elementos, las relaciones y los diagramas.

- **Los *elementos*** son abstracciones que actúan como unidades básicas de construcción. Hay cuatro tipos, los *estructurales*, los de *comportamiento*, los de *agrupación* y los de *notación*. En cuanto a los elementos estructurales son las partes estáticas de los modelos y representan aspectos conceptuales o materiales. Los elementos de comportamiento son las partes dinámicas de los modelos y representan comportamientos en el tiempo y en el espacio. Los elementos de agrupación son las partes organizativas de UML, establecen las divisiones en que se puede fraccionar un modelo. Sólo hay un elemento de agrupación, el paquete, que se emplea para organizar otros elementos en

grupos. Los elementos de notación son las partes explicativas de UML, comentarios que pueden describir textualmente cualquier aspecto de un modelo. Sólo hay un elemento de notación principal, la nota (ARREGUI, 2004).

- **Las *relaciones*** son abstracciones que actúan como unión entre los distintos *elementos*. Hay cuatro tipos, la *dependencia*, la *asociación*, la *generalización* y la *realización* (ARREGUI, 2004).

- **Los *diagramas*** son la disposición de un conjunto de elementos, que representan el sistema modelado desde diferentes perspectivas. UML tiene nueve diagramas fundamentales, agrupados en dos grandes grupos, uno para modelar la estructura estática del sistema y otro para modelar el comportamiento dinámico. Los **diagramas estáticos son:** el de *clases*, de *objetos*, de *componentes* y de *despliegue*. Los **diagramas de comportamiento son:** el de *Casos de Uso*, de *secuencia*, de *colaboración*, de *estados* y de *actividades* (ARREGUI, 2004).

3.4. MÉTRICA VALIDACIÓN DE SOFTWARE ISO-9126

ISO 9126 es un estándar internacional para la evaluación de la calidad del software. Está reemplazado por el proyecto Suaré, ISO 25000:2005, el cual sigue los mismos conceptos.

El estándar está dividido en cuatro partes las cuales dirigen, realidad, métricas externas, métricas internas y calidad en las métricas de uso y expendido. El modelo de calidad establecido en la primera parte del estándar, ISO 9126-1, clasifica la calidad del software en un conjunto estructurado de características

y sub-características de la siguiente manera:

Funcionalidad - Un conjunto de atributos que se relacionan con la existencia de un conjunto de funciones y sus propiedades específicas. Las funciones son aquellas que satisfacen las necesidades implícitas o explícitas.

Fiabilidad - Un conjunto de atributos relacionados con la capacidad del software de mantener su nivel de prestación bajo condiciones establecidas durante un período establecido.

Usabilidad - Un conjunto de atributos relacionados con el esfuerzo necesario para su uso, y en la valoración individual de tal uso, por un establecido o implicado conjunto de usuarios.

Eficiencia - Conjunto de atributos relacionados con la relación entre el nivel de desempeño del software y la cantidad de recursos necesitados bajo condiciones establecidas.

Mantenibilidad - Conjunto de atributos relacionados con la facilidad de extender, modificar o corregir errores en un sistema software.

Portabilidad - Conjunto de atributos relacionados con la capacidad de un sistema de software para ser transferido y adaptado desde una plataforma a otra.

Calidad en uso - Conjunto de atributos relacionados con la aceptación por parte del usuario final y Seguridad.

CAPITULO IV

PRESENTACIÓN DE RESULTADOS

4.1. ANÁLISIS Y DISEÑO DE LA APLICACIÓN UTILIZANDO SCRUM

Las principales razones del uso de un ciclo de desarrollo iterativo e incremental de tipo SCRUM para la ejecución de este proyecto son:

Sistema modular. Las características del sistema interconectividad basado en api rest en aplicaciones de la municipalidad provincial de lampa permiten desarrollar una base funcional mínima y sobre ella ir incrementando más funcionalidades o modificando el comportamiento o apariencia de las ya implementadas

Para entender todo el proceso de desarrollo del SCRUM se describirá de forma general las fases y los roles Scrum que se realizaron en el presente proyecto, se puede dividir de forma general en 3 fases, que podemos entender como reuniones.

Las reuniones

La planificación del Backlog. Se realizó una reunión previa de con el gerente

de desarrollo humano de la municipalidad provincial de Lampa en el cual se hice una descripción general de cómo funciona el área administrativa y en la cual nos indica los requerimientos y las prioridades del sistema.

En esta fase también se definió la planificación del **"sprint 0"**, en la que se decidió los objetivos reducir el tiempo en los procesos administrativos de modo que estos sean más eficientes, homogenizar los datos en un solo repositorio de modo que los datos sean fáciles de usar por los administradores del sistema también se discutió el trabajo que hay que realizar para esa iteración.

Los roles

Dentro de esto tenemos

Las personas comprometidas con el proyecto y proceso del SCRUM

- **Product Owner: T**uvimos al señor Marco Corrales quien es la persona encargo que las funciones administrativas fueran más eficientes.
- **SCRUM master:** El señor Dilmerd Atencio Flores quien fue el encargado de comprobar que el modelo y la metodología funcionara.
- **Team (equipo de desarrollo):** Ya que el proyecto fue realizado por dos personas tenemos al Sr. Dilmerd Atencio Flores y al Sr. David Mamani Machaca quienes se hicieron cargo del diseño y análisis del sistema los cuales tienen autoridad para organizar y tomar decisiones para conseguir su objetivo.

Aunque no son parte del proceso del scrum son necesarios para la retroalimentación del proceso y asi poder revisar y planear cada sprint

- **Usuarios:** Las personas que trabajan en la municipalidad
- **Stakeholders:** Fueron los administradores que pruebas sistema para probarlo y medir su eficiencia

Los elementos del SCRUM

- **Product Backlog:** Los cuales detallamos más adelante en el presente proyecto.
- **Sprint Backlog:** Lista de tareas que realizamos en un “sprint”. Detallados más adelante

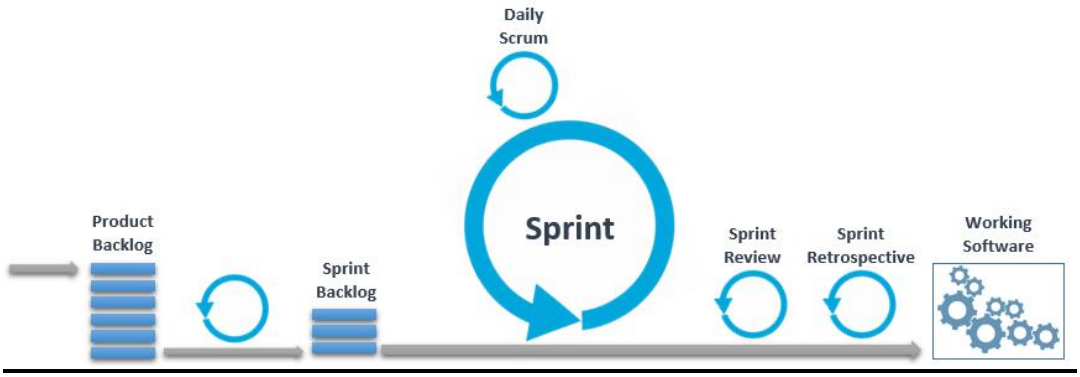

Figura 8: Ciclo de desarrollo SCRUM

4.1.1. PLANIFICACIÓN (SPRINT 0)

Lo primero que tuvimos que hacer al desarrollar el proceso de desarrollo, es la planificación; esta etapa al igual que el resto de etapas fueron analizadas durante el desarrollo de cada Sprint, Precisamente con el fin de iniciar con las iteraciones, realizamos una definición global de las iteraciones.

Este análisis inicial se denomina “Sprint 0”. Para definir esta realizamos la definición de actividades, haciendo un análisis preliminar de los procesos Figura 9, para entender los alcances del presente proyecto, realizando una estimación de tiempos y recursos a utilizar.

4.1.2. ANÁLISIS DE PROCESOS

Para definir los alcances funcionales, se tuvo que analizar los procesos ya definidos internamente por la municipalidad provincial de Lampa de la cual realizamos el presente diagrama de procesos

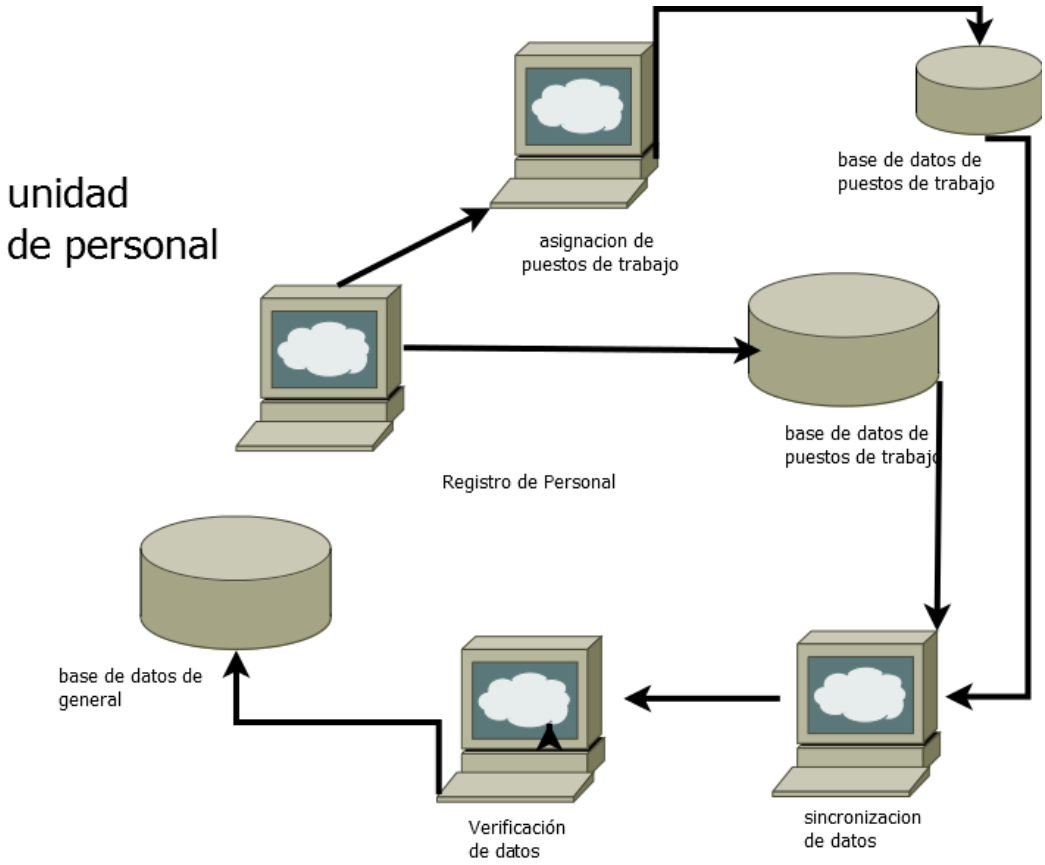

Figura 9: proceso de asignación de personal

Macro Proceso – PROCESO ADMINISTRATIVO DE LA OFICINA GENERAL DE ADMINISTRACIÓN de la municipalidad provincial de Lampa el cual contiene un conjunto de procesos tales como: Unidad de Recursos Humanos RR., Unidad de abastecimiento, Unidad de bienes patrimoniales, Unidad de contabilidad y la unidad de tesorería.

Proceso: Unidad de Recurso Humanos administra al personal, (obreros y empleados) registrando el tipo de personal, sus deberes, monitoreando el

cumplimiento de sus funciones

En la figura 10 procesos de administración de personal serán descompuestos en diagramas que permitieron analizar al detalle los subprocesos que son los siguientes

- **Registro de personal:** Este sub proceso permite el manejo los datos proporcionados por el personal tales como actualizar los datos, eliminarlos de ser necesarios, o ingresar nuevos datos de personal entrante
- **Asignación de Puesto de trabajo:** Este sub proceso indica en qué lugar del municipio presta servicios el personal asignado.
- **Sincronización de datos:** El cual provee de los datos actualizados a las Unidades que requieran de los datos del personal.
- **Control y verificación:** El cual provee de los datos de ingreso y salida del personal y verifica que estos estén en sus puestos de trabajo asignados.

4.1.3 DIAGRAMA DE PROCESOS

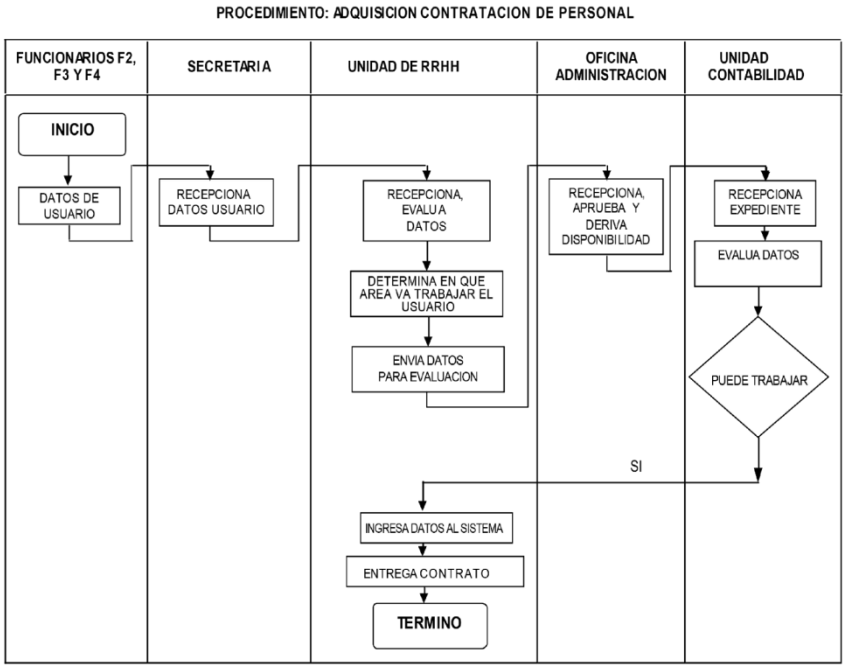

Figura 10: DIAGRAMA DE PROCESOS

De los procesos descritos en el diagrama se han identificado los requerimientos funcionales y no funcionales que debe cumplir el sistema a implementar. Los cuales se encuentran descritos en las siguientes tablas:

REQUISITO FUNCIONAL 1

Tabla 1: Requisito funcional 1

Numero de requisito	**RF 1**
Nombre del requisito	Registrar Administrador con su respectiva contraseña
Tipo	Requisito
Descripción	Necesarias para que pueda administrar el sistema.
Prioridad	Alta

Fuente: Elaboración propia

REQUISITO FUNCIONAL 2

Tabla 2: Requisito funcional 2

Numero de requisito	RF 2
Nombre del requisito	Funcionamiento del sistema
Tipo	Requisito
Descripción	La aplicación deberá poder utilizarse con cualquier navegador web.
Prioridad	Alta

Fuente: Elaboración propia

REQUISITO FUNCIONAL 3

Tabla 3: Requisito funcional 6

Numero de requisito	RF 3
Nombre del requisito	Alerta del sistema
Tipo	Requisito
Descripción	Enviará una alerta al administrador cuando ocurra alguno de los siguientes eventos: Registro de nueva cuenta, 2 o más intentos fallidos en el ingreso de la contraseña de administrador.
Prioridad	Alta

Fuente: Elaboración propia

REQUISITO FUNCIONAL 4

Tabla 4: Requisito funcional 4

Numero de requisito	RF 4
Nombre del requisito	Controlar los privilegios del administrador
Tipo	Requisito
Descripción	El sistema debe poder controlar los niveles de acceso para poder asegurar los datos de los usuarios (no poder eliminar un usuario sin previa autorización del gerente).
Prioridad	Alta

Fuente: Elaboración propia

REQUISITO FUNCIONAL 5

Tabla 5: Requisito funcional 5

Numero de requisito	RF 5
Nombre del requisito	Controlar los privilegios del administrador
Tipo	Requisito
Descripción	El sistema debe poder controlar los niveles de acceso para poder asegurar los datos de los usuarios (no poder eliminar un usuario sin previa autorización del gerente).
Prioridad	Alta

Fuente: Elaboración propia

REQUISITO FUNCIONAL 6

Tabla 6: Requisito funcional 6

Numero de requisito	RF 6
Nombre del requisito	Verificar si el trabajador tiene datos correctos
Tipo	Requisito
Descripción	Para poder controlar si el trabajador está registrado en la RENIEC validando sus datos.
Prioridad	Alta

Fuente: Elaboración propia

REQUISITO FUNCIONAL 7

Tabla 7: Requisito funcional 7

Numero de requisito	RF 7
Nombre del requisito	Consultar datos del trabajador
Tipo	Requisito
Descripción	Para poder verificar si el trabajador está inscrito en el sistema
Prioridad	Media

Fuente: Elaboración propia

REQUISITO FUNCIONAL 8

Tabla 8: Requisito funcional 8

Numero de requisito	RF 8
Nombre del requisito	Consultar o modificar categoría del trabajador
Tipo	Requisito
Descripción	Necesario para visualizar o modificar la categoría del trabajador.
Prioridad	Media

Fuente: Elaboración propia

REQUISITO FUNCIONAL 9

Tabla 9: Requisito funcional 9

Numero de requisito	RF 9
Nombre del requisito	Identificar quien ingreso al trabajador
Tipo	Requisito
Descripción	Identifica quien realizo el alta del trabajador en el sistema
Prioridad	Alta

Fuente: Elaboración propia

Requisito funcional 10

Tabla 10: Requisito funcional 10

Numero de requisito	RF 10
Nombre del requisito	Asignar lugar de trabajo del trabajador
Tipo	Requisito
Descripción	Asigna en que puesto de trabajo se desempeña el trabajador
Prioridad	Alta

Fuente: Elaboración propia

REQUISITO FUNCIONAL 11

Tabla 11: Requisito funcional 11

Numero de requisito	RF 11
Nombre del requisito	Asistencia del trabajador al centro laboral
Tipo	Requisito
Descripción	Registrará los datos de ingreso y salida del trabajador del centro laboral
Prioridad	Alta

Fuente: Elaboración propia

REQUISITO FUNCIONAL 12

Tabla 12: Requisito funcional 12

Numero de requisito	RF 12
Nombre del requisito	Pagar al trabajador
Tipo	Requisito
Descripción	El sistema ingresara el monto a pagar al trabajador según la asistencia del trabajador.
Prioridad	Alta

Fuente: Elaboración propia

REQUISITO FUNCIONAL 13

Tabla 13: Requisito funcional 11

Numero de requisito	RF 13
Nombre del requisito	Guardar datos del trabajador
Tipo	Requisito
Descripción	El sistema guardara los datos de los usuarios en una base de datos.
Prioridad	Alta

Fuente: Elaboración propia

REQUISITO FUNCIONAL 11

Tabla 14: Requisito funcional 11

Numero de requisito	RF 14
Nombre del requisito	Realizar reporte mensual de transacciones de usuarios
Tipo	Requisito
Descripción	El sistema realizara reportes mensuales de los procesos realizados por los usuarios
Prioridad	media

Fuente: Elaboración propia

Nota: la presente lista de requerimientos fue socializada por los especialistas del área de Recursos humanos.

Los módulos identificados tentativamente durante el análisis son los siguientes:

1. **Módulo de Administración**
2. **Módulo de Gestión de datos**
3. **Módulo de Reportes**

4.1.4 ALCANCE DEL SOFTWARE

Por medio del análisis de los procesos y las funcionalidades identificadas, el software a desarrollarse tiene el siguiente alcance.

- **Módulo de administración.** El cual permite Crear, eliminar, modificar administradores para poder controlar el acceso a los diferentes módulos del sistema desarrollado para la municipalidad de Lampa.
- **Módulo de Gestión** El cual permite Cargar y procesar información de los trabajadores del municipio para el uso de los datos del mismo por las diferentes unidades de administración
- **Reporte mensual de datos procesados** Realiza un reporte mensual de

todos los tramites hechos por el sistema realizados en el municipio

4.1.5. CONFORMACIÓN DEL EQUIPO DE TRABAJO

EQUIPO DE TRABAJO Y ROLES

Tabla 15: Equipo de trabajo y Roles

ROL	PERSONA	ÁREA
Product owner	Marco Corrales	Jefe de oficina de administración
SCRUM Master	Dilmerd Atencio	Análisis – desarrollo
TEAM (equipo de desarrollo)	David Mamani	Análisis - desarrollo

4.1.6 DEFINICIÓN DEL BACKLOG DEL PRODUCTO

Usando el análisis previo de los requerimientos funcionales del sistema obtuvimos **El backlog del producto** es básicamente el listado de ítems que resultaron del análisis previo “Sprint0” de los cuales se tiene

EL Backlog del producto para el presente proyecto se encuentra definido por la siguiente tabla.

Tabla 16:Backlog Product

id	nombre	importancia	Tiempo estimado (semanas)	Comentarios
1	Módulo de administración del sistema	800	5	Análisis y desarrollo de las funcionalidades para la administración de personal
2	Módulo de gestión de personal	900	5	

Fuente: Elaboración propia

La importancia está estimada de acuerdo a las necesidades del **producto owner** y está cuantificado con números enteros entre 0 y 1000, de acuerdo a la siguiente tabla

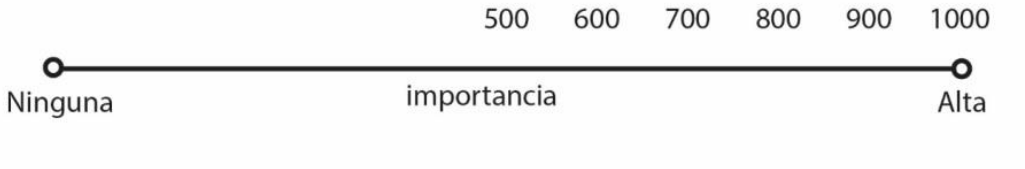

Figura 11: Escala de importancia definida por Product Owner

4.2 DISEÑO

4.2.1 ARQUITECTURA GENERAL DEL SISTEMA

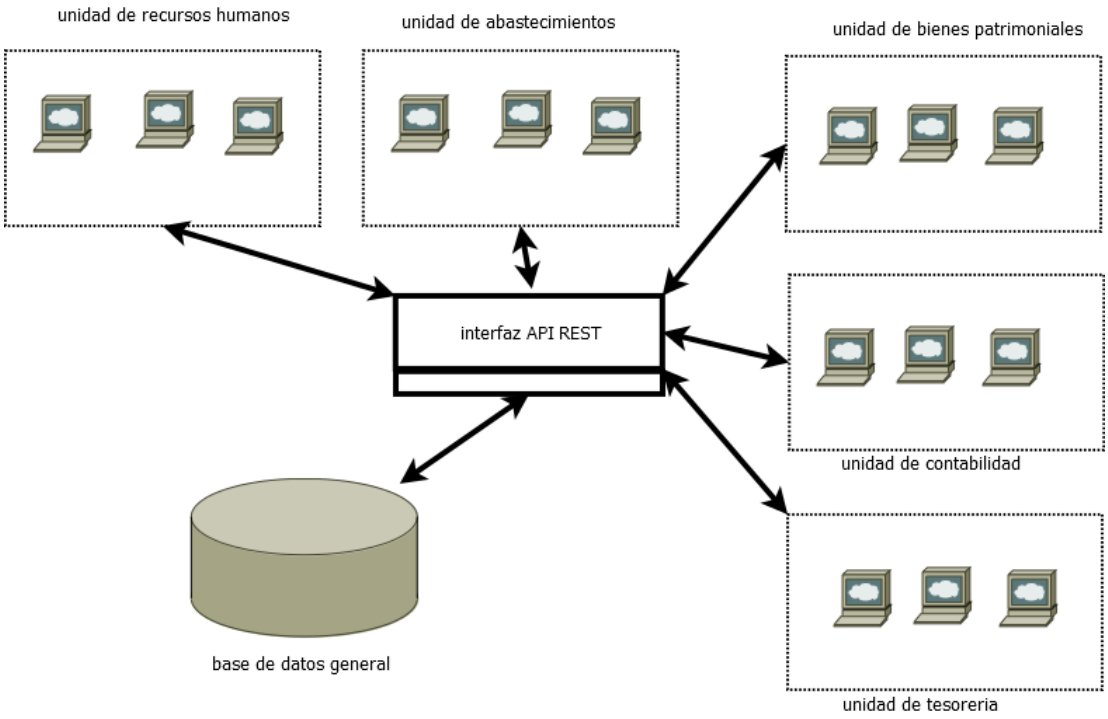

Figura 12: interfaz del sistema usando API REST

Fuente: Elaboración propia

Para la implementación del sistema, se utilizará un modelo de N capas distribuidas de la siguiente manera

- **Capa de presentación:** Está representada por el entorno grafico o GUI que es la presentación del sistema para su manejo, la cual está

desarrollada en un PHP, para que se pueda acceder en cualquier sistema operativo ya que solo necesita para su funcionamiento un browser web

- **Capa de negociación:** En esta parte desarrollamos un API REST usando JSON para la manipulación de los datos el cual se comunica con una API del RENIEC para poder validar los datos de las personas, aquí es donde todas las reglas del sistema que deben aplicarse. Esta capa se comunica con la capa de presentación (entorno gráfico) y con la base de datos (REPOSITORIO) para solicitar al gestor de base de datos almacenar o recuperar datos del sistema
- **Capa de datos:** Nuestra base de datos creada a partir del análisis que hicimos del sistema llamado también repositorio donde residen los datos y es la encargada de acceder a los mismos, en nuestro caso solo usamos una base de datos **centralizado** para evitar la redundancia de los mismo.

4.2.2 SPRINT1 “módulo de administración”

El primer sprint tiene como objetivo implementar las funciones requeridas para la administración de los recursos utilizados por el sistema: esto es crear, eliminar, modificar usuarios y equipos para controlar el ingreso al sistema.

4.2.2.1 DIAGRAMAS DE CASOS DE USOS

Tabla 17: Historias de usuario para el sprint

ID	HISTORIA DE USUARIO	IMPORTANCIA DEL PRODUCT OWNER	IMPORTANCIA TÉCNICA	DESCRIPCIÓN
1	Creación de perfiles de administrador	800	1000	La administración de las personas encargadas del sistema es muy importante dado que el acceso debe estar restringido
2	Iniciar sesión en el sistema	800	1000	Consiste en brindad seguridad de acceso a la aplicación
3	Administración de los usuarios	1000	800	El sistema debe administrar a los usuarios (empleados y obreros) para que estos puedan ingresar ser usados por las demás unidades del municipio

Fuente: Elaboración propia

4.2.3 ANALISIS

Están identificados las historias de usuarios, los actores y se diagramó los casos de uso para la implementación del Sprint1.

4.2.4 Actores Del Sistema

Tabla 18: Identificación del actor

ACTOR	DESCRIPCIÓN
Administrador	Este usuario tiene privilegios para administrar y gestionar los recursos usados por el sistema.

Fuente: Elaboración propia

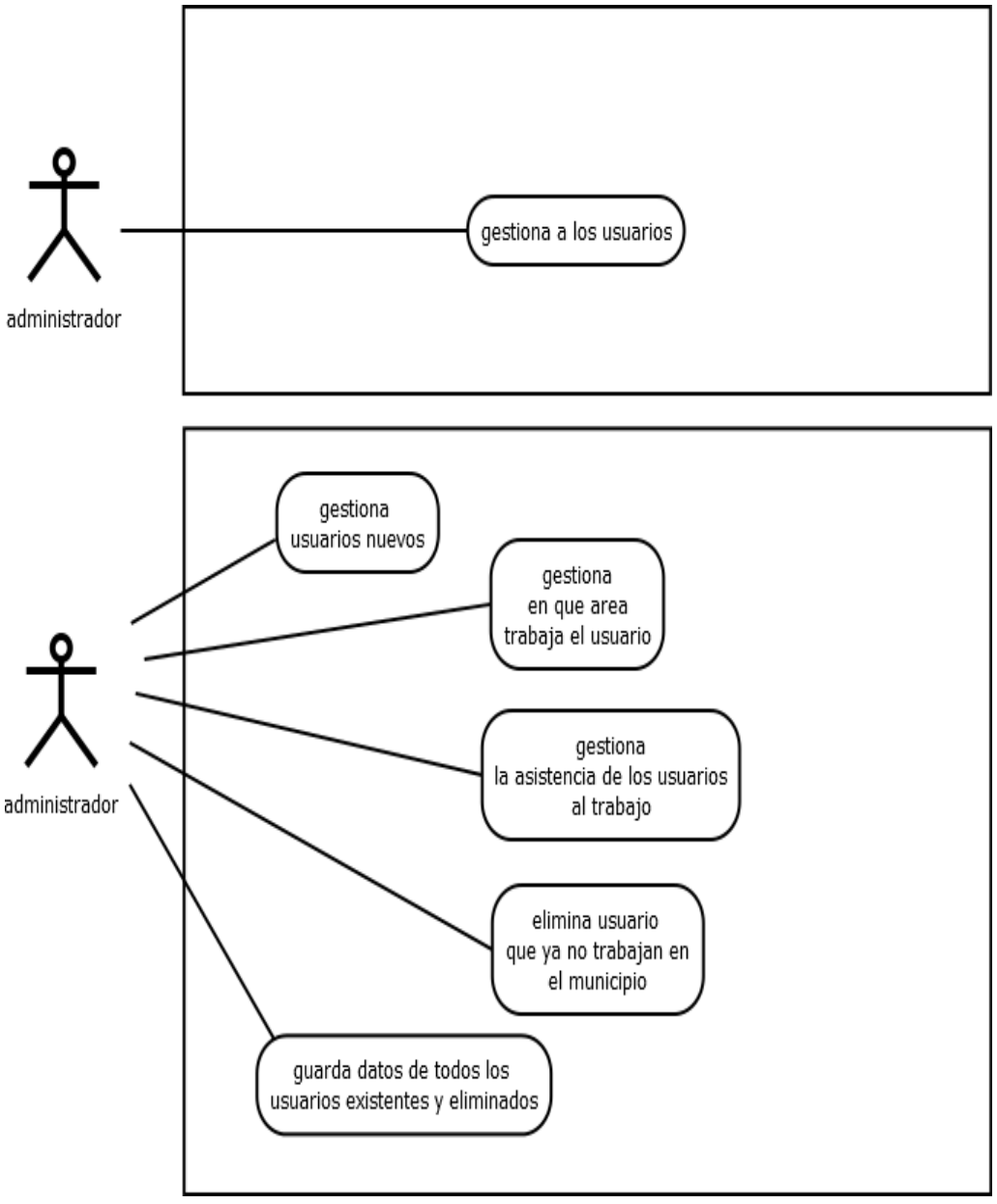

Figura 13: Casos de uso administración

Fuente: Elaboración propia

4.2.5 ESPECIFICACIÓN DE LOS CASOS DE USO

Tabla 19: Especificaciones caso de uso: Gestionar Administradores

Nombre del caso de uso	Iniciar sesión en el sistema
Actores	Administrador/es
Propósito	Ingresar al sistema
Tipo	Principal
Requerimiento	Tener un código de usuario y una clave
Precondiciones	El administrador debe estar registrado para poder ingresar
FLUJO NORMAL DE EVENTOS	
Acción del actor	**Respuesta del sistema**
1. Se inicia cuando el administrador entra en la página web del sistema y aprieta ingresar como administrador 2. El administrador ingresa su código de usuario y clave de acceso	3. El sistema muestra la pantalla de ingreso al sistema 4. El administrador ingresa al sistema
Flujo alterno 5. Si la clave o el código de usuario está equivocado enviara un mensaje y se registrara en el sistema de reportes.	

Tabla 20: Especificaciones caso de uso: Gestionar usuarios

Nombre del caso de uso	Ingresa nuevo trabajador al sistema
Actores	Administrador
Propósito	Ingresar al sistemas nuevos usuarios (trabajadores)
Tipo	Principal
Requerimiento	Tener nombre, apellidos, código y categoría del trabajador
Precondiciones	El administrador tiene que haber iniciado sesión El trabajador nuevo no tiene que estas inscrito en el repositorio El administrador debe estar registrado en el repositorio
FLUJO NORMAL DE EVENTOS	
Acción del actor	**Respuesta del sistema**
1. El administrador ingresa datos el nuevo trabajador Nombre: Apellido Categoría 6. El administrador acepta el nuevo código de usuario	2. El sistema muestra la pantalla de ingreso de datos 3. El sistema revisa que los datos sean reales validando su información con la del RENIEC 4. El sistema verifica que el usuario no se encuentre registrado 5. El sistema crea un código de identificación de usuario 7. El sistema almacena la información en el REPOSITORIO 8. Muestra información registrada del trabajador
Flujo alterno 2. El trabajador ya se encuentra registrado	3. El sistema manda un mensaje de error “el trabajador ya existe”.

Tabla 21: Especificaciones caso de uso: gestionar lugar de trabajo del trabajador

Nombre del caso de uso	Indicar puesto de trabajo del usuario (trabajador)
Actores	Administrador/es
Propósito	Indica el lugar de labor asignado al trabajador
Tipo	Principal
Requerimiento	Los datos del lugar donde va trabajar
Precondiciones	El administrador debe estar registrado para poder ingresar los datos
FLUJO NORMAL DE EVENTOS	
Acción del actor	**Respuesta del sistema**
1. El administrador entra en la página web del sistema y aprieta el módulo de gestionar lugar de trabajo 3. El administrador ingresa datos del lugar asignado al trabajador. Cargo Oficina Categoría 6. El administrador acepta la información actualizada	2. El sistema muestra la pantalla de lugar de trabajo 4. El sistema almacena la información en el REPOSITORIO 5. Muestra información actualizada del trabajador
Flujo alterno El administrador puede salir del sistema sin guardar	

Tabla 22: Especificaciones caso de uso: asistencia del usuario

Nombre del caso de uso	Asistencia del usuario
Actores	Administrador/es
Propósito	Registra la asistencia de los trabajadores
Tipo	Principal
Requerimiento	Información de las asistencias del trabajador
Precondiciones	El administrador debe haber iniciado sesión El trabajador debe estar registrado
FLUJO NORMAL DE EVENTOS	
Acción del actor	**Respuesta del sistema**
1. Ingresa al módulo administrar la asistencia de los trabajadores 3. El administrador ingresa código de trabajador realizar la búsqueda de las actividades	2. El sistema muestra la pantalla del módulo de asistencia 4. El sistema busca los datos requeridos del trabajador haciendo una búsqueda en el repositorio 5. El sistema envía un reporte al administrador.
Flujo alterno 6. Si el usuario no existe “el sistema envía un mensaje de error”	

Tabla 23: Especificaciones caso de uso: eliminar usuario

<table>
<tr><td>Nombre del caso de uso</td><td colspan="2">Eliminar usuario</td></tr>
<tr><td>Actores</td><td colspan="2">Administrador/es</td></tr>
<tr><td>Propósito</td><td colspan="2">Eliminar un usuario del sistema</td></tr>
<tr><td>Tipo</td><td colspan="2">Principal</td></tr>
<tr><td>Requerimiento</td><td colspan="2">Nombre y código del trabajador</td></tr>
<tr><td>Precondiciones</td><td colspan="2">Haber ingresado como administrador
El trabajador debe de estar inscrito en el sistema</td></tr>
<tr><td colspan="3">FLUJO NORMAL DE EVENTOS</td></tr>
<tr><td colspan="2">Acción del actor</td><td>Respuesta del sistema</td></tr>
<tr><td colspan="2">1. El administrador ingresa al módulo eliminar trabajadores
3. El administrador ingresa su código de usuario y clave de acceso del trabajador para poder eliminarlo
5. Confirma la eliminación del trabajador</td><td>2. El sistema muestra la pantalla de eliminar usuarios
4. El sistema envía un mensaje “ESTA SEGURO QUE QUIERE ELIMINAR AL TRABAJADOR”
6. El sistema cambia el estado del trabajador y lo envía los datos al REPOSITORIO no elimina los datos elimina al usuario solo cambia el estado del mismo como no trabajador
7. Envía mensaje de acción realizada</td></tr>
<tr><td colspan="3">Flujo alterno
8. El sistema hace un reporte de los datos realizados al Gerente</td></tr>
</table>

4.2.7 Modelo de datos

Del análisis de los casos de uso se determina la necesidad de utilizar el modelo de datos descrito en el siguiente diagrama de entidad relación. Los mismos que se utilizaran para realizar la base de datos del sistema

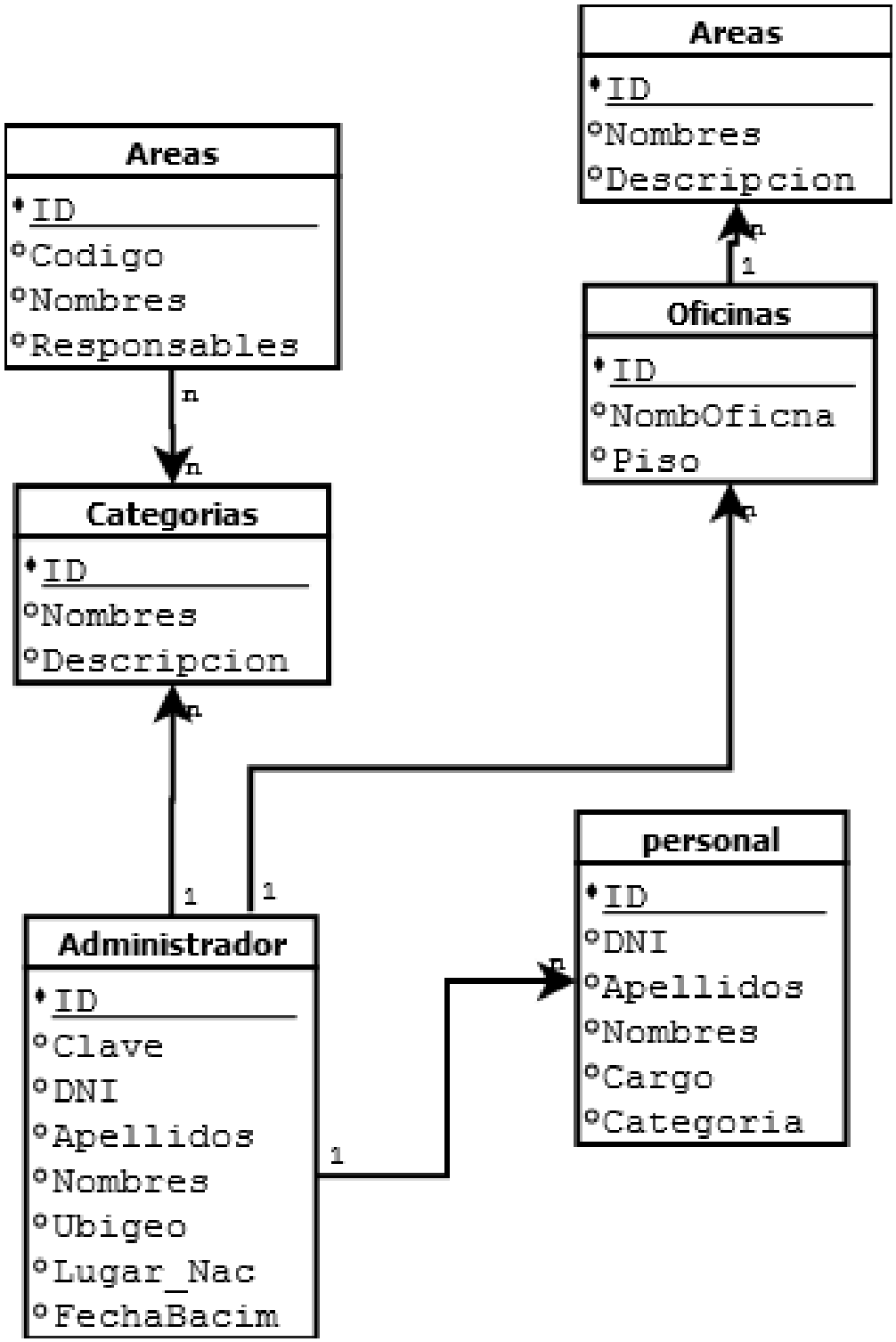

Figura 14: Modelo de entidad relación

Fuente: Elaboración propia

4.3 MODELAMIENTO DEL API REST DESARROLLADO

4.3.1. DIAGRAMA DE SECUENCIAS

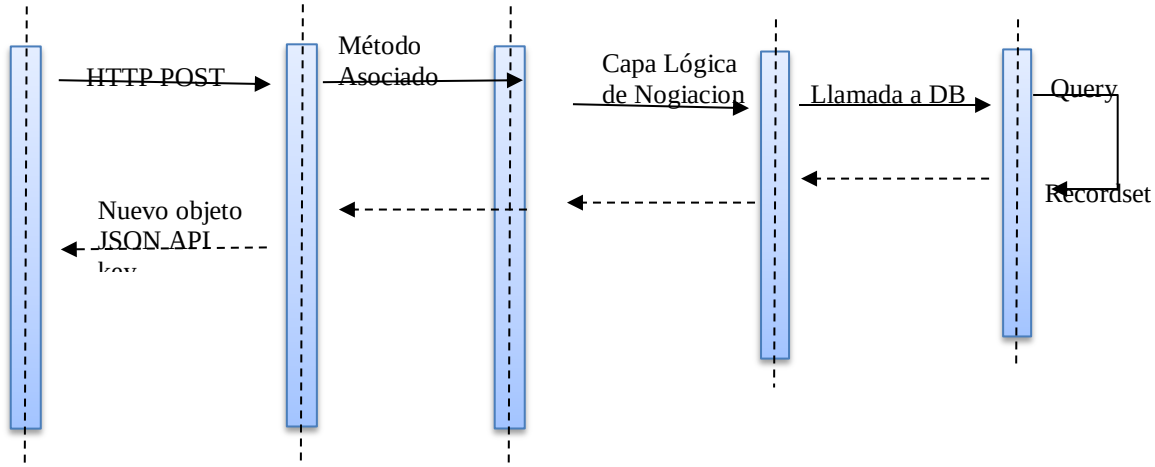

Figura 15: diagrama de secuencias

Fuente: Elaboración propia

La secuencia inicia en el pedido mediante una petición HTTP mediante el método POST que ayuda a proteger la información de los argumentos, así como poder enviar una gama más grande de datos.

Se tiene que el REQUEST usa el método POST para él envió de datos en lugar del método GET esto para mantener la privacidad y seguridad de los datos, siendo una configuración de desarrollo saliendo del esquema general de peticiones GET en la que están implementadas la mayoría de las API REST.

En el caso de la implementación se envía una estructura encapsulada y cifrada la secuencia que sigue el Request HTTP, se mostraría de la siguiente forma:

4.3.2. DIAGRAMA DE ENTIDAD RELACIÓN

El diagrama de entidad relación muestra la relación directa entre los componentes que intervienen en el diseño. Desde la petición HTTP POST, con los argumentos que se envían para realizar el pedido al servidor HTTP que contiene el API REST en modo espera

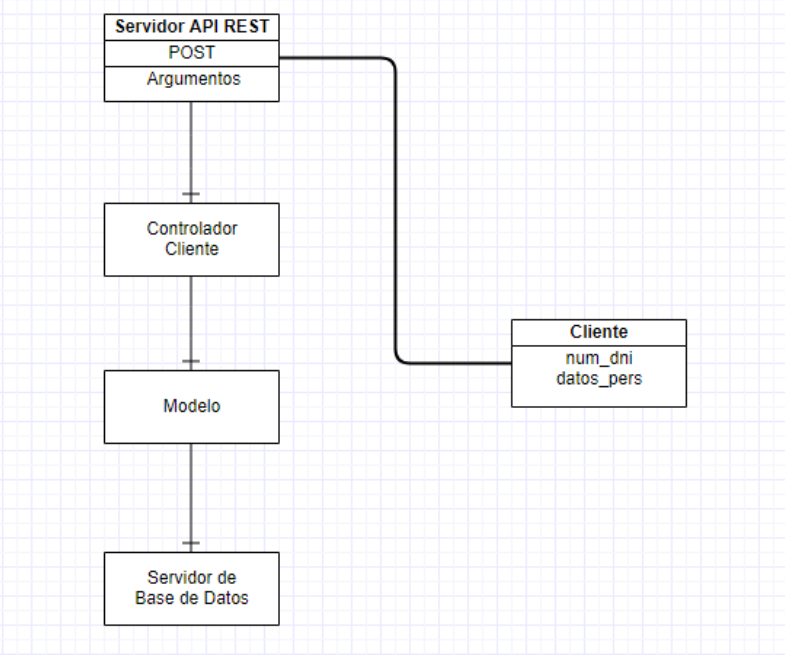

Figura 16: diagrama de entidad relación

4.3.3. DIAGRAMA DE ACTIVIDADES

Aquí se describe las actividades del cliente sobre el servidor REST, cada una de las acciones y valores obtenidos, además de la monitorización de administrador para verificar los errores, así como las peticiones hechas durante un lapso y la integridad de las consultas y los resultados obtenidos a través del log o historial de actividades.

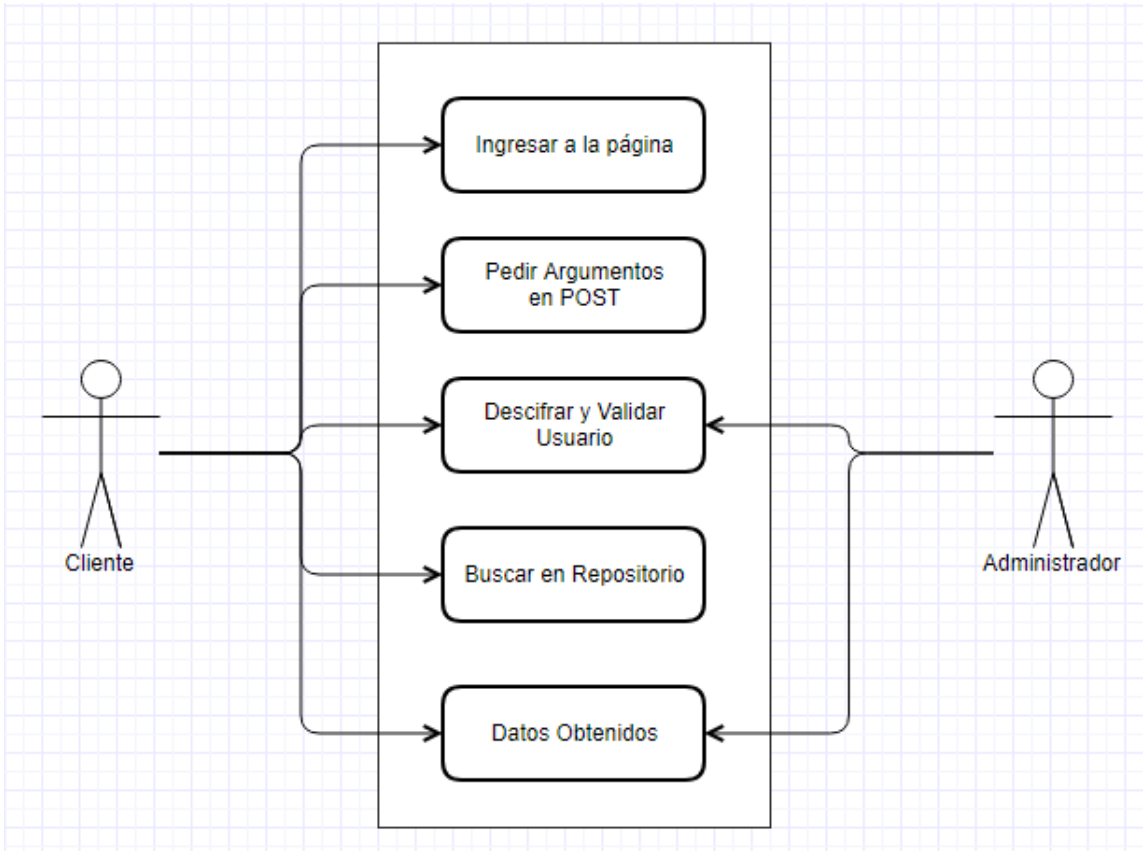

Figura 17: Diagrama de actividades

4.3.4. DIAGRAMA DE INTERACCIÓN

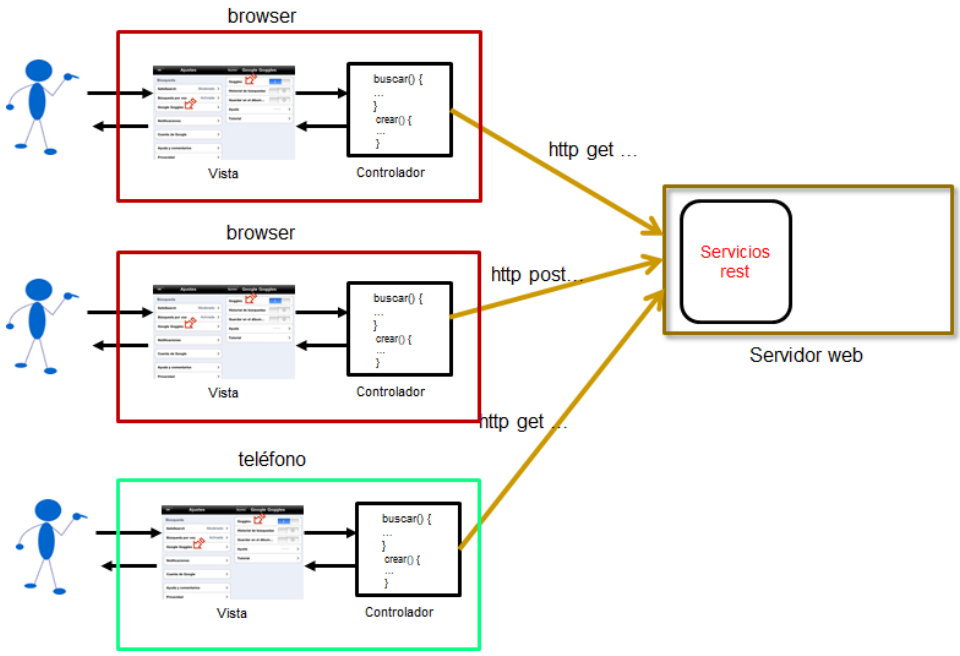

Figura 18: diagrama de iteración

En el presente diagrama vemos como se hace la interacción web del sistema por parte de los administradores del municipio haciendo uso del API que hemos creado.

4.3.5. INTERFAZ WEB

Figura 19: Interfaz web

Se puede observar la interfaz web diseñada para la consulta interna o por API REST, la conmutación se lleva por medio de la REQUEST del protocolo HTTP la interfaz que despliega el contenido web o HTML, sobre la que directamente emite un contenido en el formato o notación de objetos JSON que es el moderno protocolo para compartir información entre servicios web, locales o industriales como el caso de twitter, Facebook e incluso google.

4.3.6 PANTALLA DE PRESENTACIÓN

Figura 20: Pantalla de presentación inicial

4.3.7 ACCESO DE USUARIO DE ÁREA

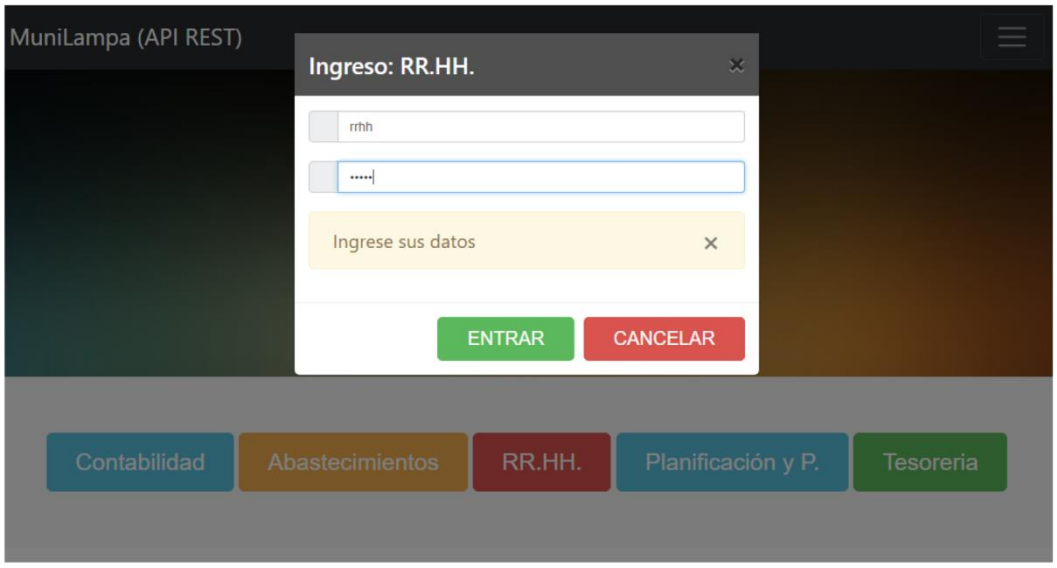

Figura 21: Pantalla de ingreso al sistema

4.3.8 BÚSQUEDAS BASADAS EN NUMERO DE DNI

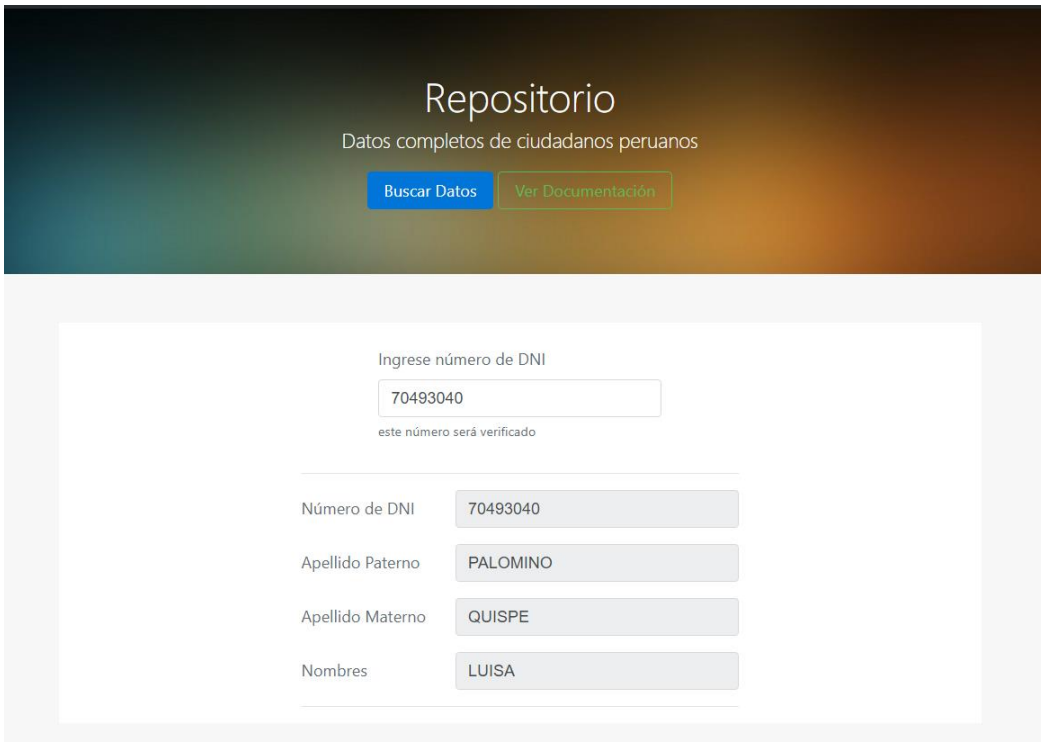

Figura 22: Ejemplo usando la interfaz web

Para realizar la búsqueda de datos se ingrese el número de DNI de un total de 8 dígitos, tenga en cuenta que la búsqueda será cancelada si no cumple con los parámetros de búsqueda, también se debe tener en cuenta que si los datos de un DNI ingresado no se muestran es por que posiblemente la persona este registrada como fallecido(a), por lo que la información se mostrara únicamente de personas activas en el repositorio de datos.

Exportación de datos en formato json

{"dni" : "70493040", "appaterno" : "PALOMINO", "apmaterno" : "QUISPE", "nombres" : "LUISA"}

Se exporta la información en formato JSON para poder manipularlo independientemente del lenguaje host, ya sea javascritp en Frameworks como NodeJS o Frameworks basados en PHP.

Tenga en cuenta que como desarrollador tercero y usuario de esta API deberá hacer una revisión de la nomenclatura de datos JSON así como realizar la manipulación de datos sobre el lenguaje que esté trabajando el módulo de conexión ya sea de consulta o de inserción de datos para el nuevo sistema de información que está desarrollando.

4.4 PRUEBAS DE EJECUCIÓN

Por consola Linux.

$ curl http://apix.munilampa.gob.pe

Por PHP

```
$peticion = base64_encode(mcrypt_encrypt(MCRYPT_RIJNDAEL_256,
    $app_key, json_encode($parametros), MCRYPT_MODE_ECB));


$api = curl_init();
curl_setopt($api, CURLOPT_URL,
            'http://apix.munilampa.gobe.pe/');
curl_setopt($api, CURLOPT_POST, TRUE);
curl_setopt($api, CURLOPT_POSTFIELDS,
            array('request'=>$peticion, 'id'=>$app_id));
curl_setopt($api, CURLOPT_RETURNTRANSFER, 1);
$resultado = curl_exec($api);
```

5.1 RESULTADOS DE LA INVESTIGACIÓN

La población estuvo constituida por un total de 18 trabajadores administrativos que trabajan en las diferentes áreas y utilizan el aplicativo web en la Municipalidad Provincial de Lampa.

Tabla 24: Número de trabajadores por área administrativa de la Municipalidad Provincial de Lampa

ÁREAS ADMINISTRATIVAS	PERSONAL
CONTABILIDAD	4
ABASTECIMIENTOS	4
RECURSOS HUMANOS	3
PLANIFICACIÓN Y PRESUPUESTO	3
TESORERÍA	4
	18

Fuente: Área de recursos Humanos

MÉTODO DE RECOPILACIÓN DE DATOS

La recopilación de los datos para el presente trabajo de investigación se realizó a través de entrevistas a cada uno de los trabajadores de las diferentes áreas administrativas de la Municipalidad Provincial de Lampa

MÉTODOS DE TRATAMIENTOS DE DATOS

Procedimiento para el proceso de trámites por área administrativa.

El procedimiento para realizar consultas y trámites en las diferentes áreas administrativas tiene una duración promedio de 3 a 4 días hábiles lo cual hace muy lenta la efectividad de mencionados trámites.

En el siguiente cuadro y grafico muestra el promedio aproximado de trámites por mes.

NÚMERO DE TRÁMITES PROMEDIO QUE SE REALIZAN EN UN MES EN LA MUNICIPALIDAD PROVINCIAL DE LAMPA

Tabla 25: Número de trámites promedio que se realizan en un mes en la municipalidad Provincial de lampa

AREAS ADMINISTRATIVAS	PERSONAL	PROM / TRAMITES
CONTABILIDAD	4	5
ABASTECIMIENTOS	4	13
RECURSOS HUMANOS	3	9
PLANIFICACION Y PRESUPUESTO	3	4
TESORERIA	4	16
	18	47

Fuente: Elaboración propia

Frecuencias

```
FREQUENCIES VARIABLES=MUN_LAMPA_TRAD
  /STATISTICS=STDDEV VARIANCE MEAN SUM
  /BARCHART FREQ
  /ORDER=ANALYSIS.
```

Tabla 26: cantidad de trámites administrativos

Estadísticos

MUNICIPALIDAD PROVINCIAL DE LAMPA PROMEDIO DE TRAMITES MÉTODO TRADICIONAL

N	Válido	1269
	Perdidos	1566
Media		3,28
Desviación estándar		1,439
Varianza		2,072
Suma		4157

Fuente: Elaboración propia

Tabla 27: cantidad promedio de trámites al mes mediante el método tradicional

MUNICIPALIDAD PROVINCIAL DE LAMPA PROMEDIO DE TRAMITES METODO TRADICIONAL

		Frecuencia	Porcentaje	Porcentaje válido	Porcentaje acumulado
Válido	CONTABILIDAD	135	4,8	10,6	10,6
	ABASTECIMIENTOS	351	12,4	27,7	38,3
	RECURSOS HUMANOS	243	8,6	19,1	57,4
	PLANIFICACIÓN Y PRESUPUESTO	109	3,8	8,6	66,0
	TESORERÍA	431	15,2	34,0	100,0
	Total	1269	44,8	100,0	
Perdidos	Sistema	1566	55,2		
Total		2835	100,0		

Fuente: Elaboración propia

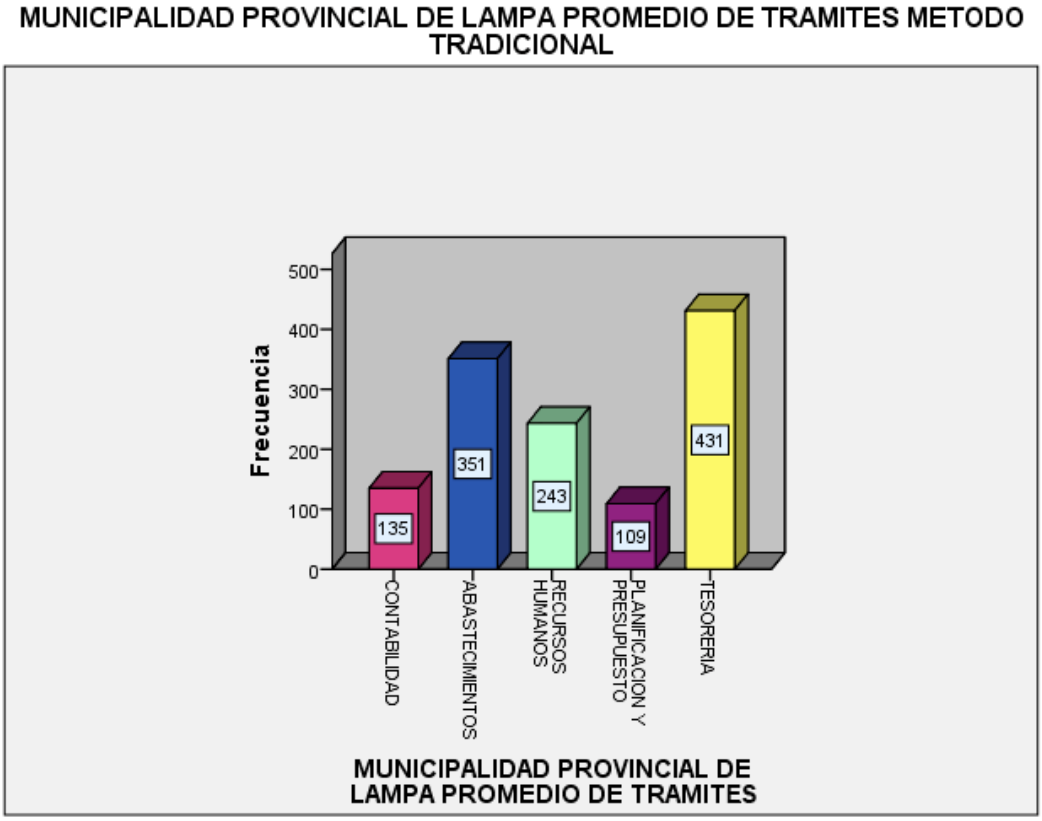

Figura 23: grafico de barras del promedio de tramites mediante el método tradicional en un mes

Tabla 28: Porcentaje de trámites promedio que se realizan en un mes en la Municipalidad de Lampa Porcentajes

AREAS ADMINISTRATIVAS	%
CONTABILIDAD	11%
ABASTECIMIENTOS	28%
RECURSOS HUMANOS	19%
PLANIFICACION Y PRESUPUESTO	9%
TESORERIA	34%

Fuente: Elaboración propia

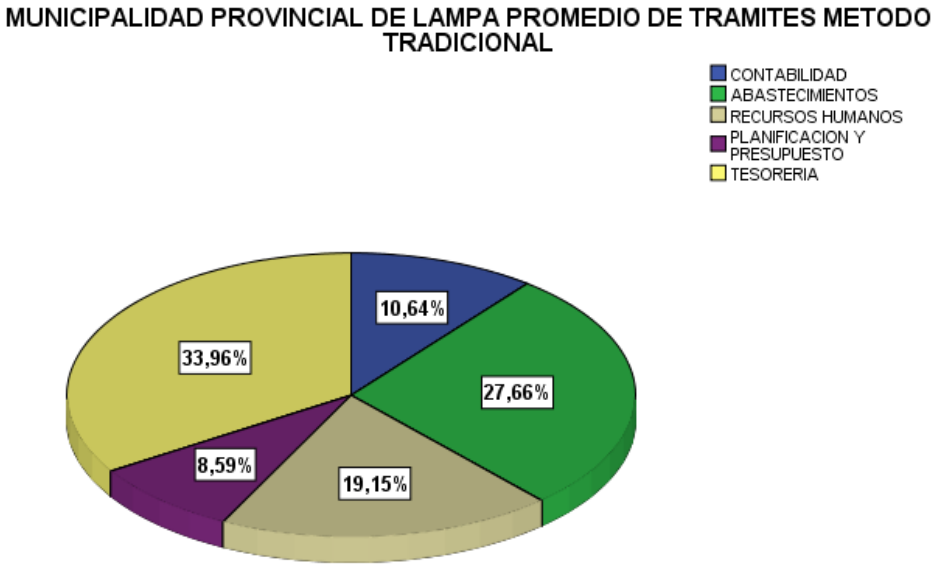

Figura 24: Grafico de Porcentaje de trámites promedio que se realizan en un mes en la Municipalidad de Lampa Porcentajes

RESULTADOS USANDO LA APLICACIÓN WEB BASADO EN EL MODELO DE COMUNICACIÓN BASADO EN API REST Y WEB SERVICE EN LA MEJORA Y COMUNICACIÓN DE LAS APLICACIONES WEB.

Tabla 29: Número de trámites promedio que se realizan en un mes en la municipalidad Provincial de lampa utilizando la aplicación web basado en API REST y Web

AREAS ADMINISTRATIVAS	PERSONAL	PROM / TRAMITES
CONTABILIDAD	4	10
ABASTECIMIENTOS	4	30
RECURSOS HUMANOS	3	20
PLANIFICACION Y PRESUPUESTO	3	7
TESORERIA	4	38
		105

Fuente: Elaboración propia

GRAFICO N° 3

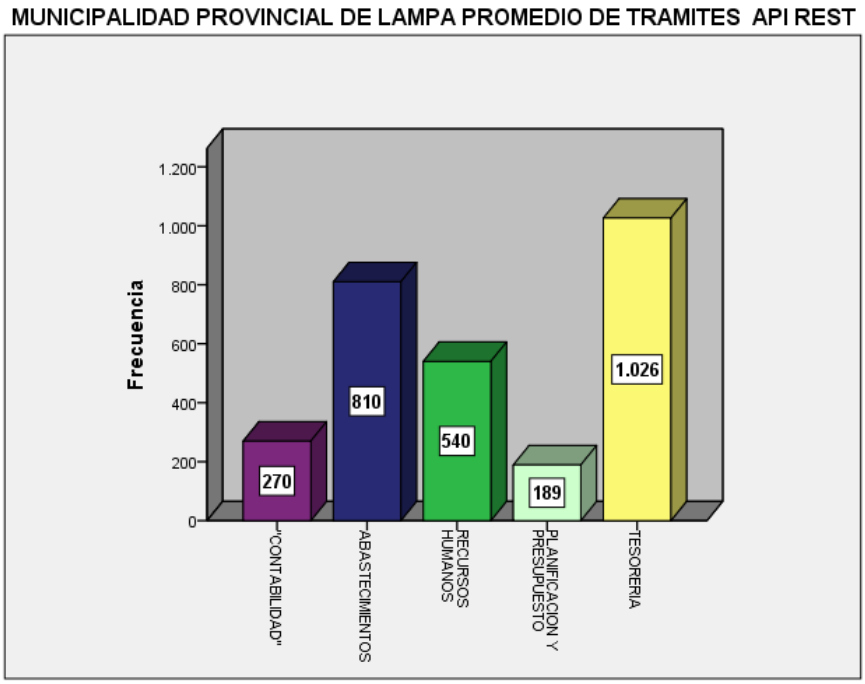

Figura 25: grafico Número de trámites promedio que se realizan en un mes en la municipalidad Provincial de lampa utilizando la aplicación web basado en API REST y Web

Tabla 30: Porcentaje de trámites promedio que se realizan en un mes en la municipalidad Provincial de lampa utilizando la aplicación web basado en API REST y Web

AREAS ADMINISTRATIVAS	%
CONTABILIDAD	8%
ABASTECIMIENTOS	20%
RECURSOS HUMANOS	15%
PLANIFICACION Y PRESUPUESTO	32%
TESORERIA	25%

Fuente: Elaboración propia

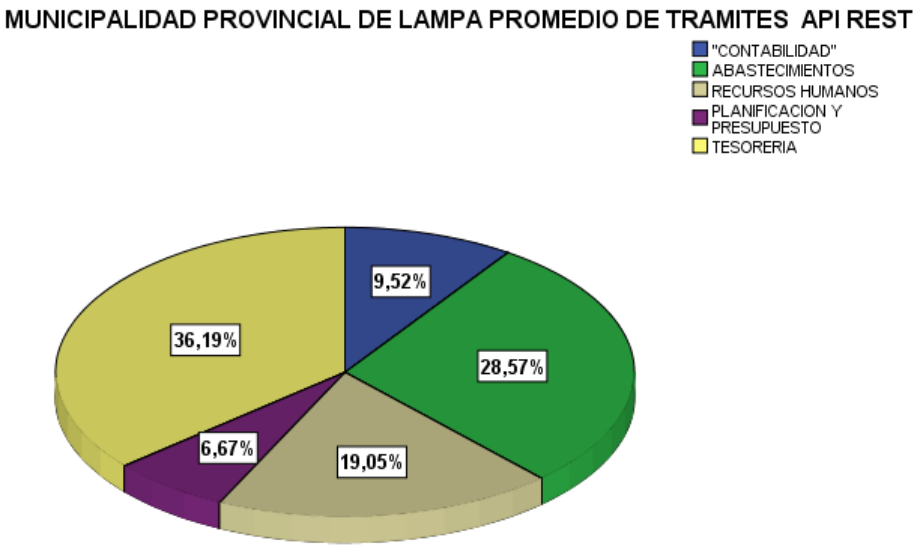

Figura 26: Grafico Porcentaje de trámites promedio que se realizan en un mes en la municipalidad Provincial de lampa utilizando la aplicación web basado en API REST y Web

CONCLUSIONES

Al concluir la investigación se ha desarrollado un modelo de comunicación Basada en API REST el cual pueden realizarse consultas basadas en HTTP, con el pack de funciones de CURL presentes en PHP e incluso la línea de comandos de sistemas operativos basados en GNU/Linux, se tiene en cuenta que la información de validación del usuario así como de la consulta tendrá un carácter cifrado para proteger el acceso únicamente a programas o desarrolladores del equipo de TIC, para la salida se hizo uso del estándar JSON para facilitar la lectura de datos, además del poder de este lenguaje de objetos para poder embeber registros únicos así como un conjunto de ellos, de forma que sea sencillo acceder y poder operar con estos datos resultantes.

Se determinó la cantidad de usuarios y el número de trámites promedio que se realizaron en un mes en las diferentes unidades administrativas de la municipalidad provincial de lampa (ver tabla 10) método tradicional de trámites administrativos, llegando a la conclusión de que existe una mejora significativa en los tramites al usar el modelo de comunicación API REST (Tabla 14)

Se logró establecer protocolos de comunicación mediante api para que se puedan comunicar con nuestro modelo de comunicación Basado en API REST el cual se puede ver en el **anexo A** de presente proyecto

RECOMENDACIONES

Se recomienda fuertemente el uso de esta WebService de interconexión de datos, para evitar la duplicidad de información en la base de datos, así como una mejor fiabilidad sobre los mismos, mejorando la imagen de la institución, así como la integridad de los datos en los diversos documentos electrónicos que estos suponen.

Se recomienda optimizar constantemente el WebService para evitar errores de datos, números de teléfono, así como implementar la consulta de mayores tramites al punto de poder enlazar datos personales con historiales de pagos, impuestos y demás para un seguimiento histórico de datos, aunque para ello se requiera mejor arquitectura de red, así como de equipos de servidores y mantenimiento constante de la oficina de tecnólogas de la información que sería la responsable.

REFERENCIAS BIBLIOGRÁFICAS

Alvarez, M. A. (2014). Ventajas e inconvenientes de API REST para el desarrollo. Retrieved from https://desarrolloweb.com/

Amodeo, E. (2013). Principios de diseño de APIs REST. *Leanpub*, *1*, 1–162. Retrieved from http://leanpub.com/introduccion_apis_rest

ARREGUI, M. (2004). TutorialUML-UP.

james Rumbaungh , Ivar Jacobson, G. B. (2000). el-lenguaje-unificado-de-modelado-manual-de-referencia.pdf.

William, E. (2014). API de los sistemas operativos, 30.

Navarro Marset, Rafael (2006). Modelado, Diseño e Implementación de Servicios Web 2006-07. ELP-DSIC-UPV.

Pautasso, Cesare; Wilde, Erik; Alarcon, Rosa (2014), REST: Advanced Research Topics and Practical Applications.

Pautasso, Cesare; Zimmermann, Olaf; Leymann, Frank (April 2008), "RESTful Web Services vs. Big Web Services: Making the Right Architectural Decision", 17th International World Wide Web Conference (WWW2008), Beijing, China.

Ferreira, Otavio (Nov 2009), Semantic Web Services: A RESTful Approach, IADIS, ISBN 978-972-8924-93.

McDonald , Mattew (2009). Creación y Diseño Web. O'Reilly Media, Inc. Edicion en Español, Ediciones ANAYA Multimedia, Fernandez Ciudad, S.L. Madrid España.

JOSEPH VILALTA,(2001). UML Guía Visual – Cómo Crear Formas de Vida Organizativa –Consultores.

ANEXOS

A

AJAX

Acrónimo de ***A****synchronous* ***J****avaScript* ***A****nd* ***X****ML* (JavaScript asíncrono y XML), es una técnica de desarrollo web para crear aplicaciones interactivas o RIA (*Rich Internet Applications*). Estas aplicaciones se ejecutan en el cliente, es decir, en el navegador de los usuarios mientras se mantiene la comunicación asíncrona con el servidor en segundo plano. De esta forma es posible realizar cambios sobre las páginas sin necesidad de recargarlas, mejorando la interactividad, velocidad y usabilidad en las aplicaciones.

API

Interfaz de programación de aplicaciones (IPA) o API (del inglés Aplicación Programming Interface) es el conjunto de funciones y procedimientos (o métodos, en la programación orientada a objetos) que ofrece cierta biblioteca para ser utilizada por otro software como una capa de abstracción. Son usadas generalmente en las bibliotecas (también denominadas “librerías").

API REST

La Transferencia de Estado Representacional (en inglés Representational State Transfer) o REST es un estilo de arquitectura software para sistemas hipermedia distribuidos como la World Wide Web. El término se originó en el año 2000, en una tesis doctoral sobre la web escrita por Roy Fielding, uno de los principales autores de la especificación del protocolo HTTP y ha pasado a ser ampliamente utilizado por la comunidad de desarrollo.

B

BASE DE DATOS

Una base de datos o banco de datos es un conjunto de datos pertenecientes a un mismo contexto y almacenados sistemáticamente para su posterior uso.

BROWSER:

(Web Browser, Navegador o visualizador) Programa que permite leer documentos en la Web y seguir enlaces (links) de documento en documento de hipertexto

C

CACHE:

Almacenamiento intermedio o temporario de información. Por ejemplo, un navegador posee un cache donde almacena las últimas páginas visitadas por el usuario y, si alguna se solicita nuevamente, el navegador mostrará la que tiene acumulada en lugar de volver a buscarla en Internet. El término se utiliza para denominar todo depósito intermedio de datos solicitados con mayor frecuencia.

CGI

(Common Gateway Interface, Interfaz Común de Intercomunicación) Conjunto de medios y formatos para permitir y unificar la comunicación entre la Web y otros sistemas externos, como las bases de datos. Similar al ActiveX.

H

HTTP

(Hypertext Transfer Protocol, Protocolo de Transferencia de Hipertexto) Es el mecanismo de intercambio de información que constituye la base funcional de la World Wide Web.

J

JSON

Acrónimo de *JavaScript Object Notation*, es un formato de texto ligero para el intercambio de datos. JSON es un subconjunto de la notación literal de objetos de JavaScript aunque hoy, debido a su amplia adopción como alternativa a XML, se considera un formato de lenguaje independiente.

JAVASCRIPT:

Lenguaje de Scripts para utilizar en páginas Web desarrollado por Netscape. Permite aumentar la interactividad y la personalización de un sitio.

L

LOGIN :

Proceso de seguridad que exige que un usuario se identifique con un nombre (user-ID o nombre de usuario) y una clave (password o contraseña), para poder acceder a una computadora o recurso.

P

PÁGINA WEB:

Es la unidad mínima de información en el WWW. Cada vez que pulsa un enlace o especifica una dirección, se carga un fichero que se le muestra en pantalla. Este fichero, llamado página, puede contener imágenes, enlaces a otras páginas, textos... puede ser tan pequeño como unas pocas líneas, o tan grande como esta (y más).

PHP

PHP, siglas recursivas en inglés de Hypertext Preprocessor (procesador de hipertexto), es un lenguaje de programación de propósito general de código del lado del servidor originalmente diseñado para el desarrollo web de contenido dinámico. Fue uno de los primeros lenguajes de programación del lado del servidor que se podían incorporar directamente en el documento HTML en lugar de llamar a un archivo externo que procese los datos.

S

SERVIDOR:

En una estructura cliente-servidor, se llama servidor a un programa que ofrece una serie de servicios, a los cuales se suele acceder por medio de programas especiales llamados clientes. Por ejemplo, a un servidor FTP, situado en un ordenador en cualquier lugar de la red se accede mediante programas FTP clientes, que son los que disponemos en nuestro ordenador. Por extensión, se suele llamar servidor también al ordenador en el que están situados estos programas.

Streaming.-

Es un término que hace referencia al hecho de escuchar música o ver vídeos sin necesidad de descargarlos, sino que se hace por fragmentos enviados secuencialmente a través de Internet.

U

URI o URL:

son las siglas en inglés de *Uniform Resource Identifier* y sirve para identificar recursos en Internet. URL son las siglas en inglés de *Uniform Resource Locator* y sirve para nombrar recursos en Internet.

W

WS

Web Services Interoperability Organization - Organización para la Interoperabilidad de Servicios Web. Su objetivo es fomentar y promover la Interoperabilidad de Servicios Web (Web Services Interoperability - WS-I) sobre cualquier plataforma, sobre aplicaciones, y sobre lenguajes de programación. Su intención es ser un integrador de estándares para ayudar al avance de los servicios web de una manera estructurada y coherente. La WS-I ha organizado los estándares que afectan a la interoperabilidad de los servicios web en una pila basada en funcionalidades.

ANEXO A
CODIGO FUENTE DEL CLIENTE

```
<?php if( !defined('LX_COREDIR') ) exit('LxCore is not present');


class Inicio extends lxApp
{
   //protected   $session;
   //protected   $db;

   public function __construct()
   {
      parent::__construct();
      // $this->session = $this->loadLib( "session" );
      //$this->loadLib( "db" ); custom lib
      //$this->loadOwn( "db" );
      $this->loadModel( "ado" );
   }

   public function index()
   {
      $this->loadView( "web/inicio" );
   }

   public function docs()
   {
   }

   public function inBuscar()
   {
      $dni = mlSecuRequest("dni");


      if( strlen($dni) != 8 ){
         echo "<br><b> DNI Inconsistente </b>";
         return;
      }

      $res = $this->apiRepository( $dni );
      $res = json_decode( $res );

      if( ! $res->success ) {
         echo "<br><b> $res->message </b>";
         return;
```

```
        }

        if( ! $res->items[0]->DNI ) {
            echo "<br><b> Sin información </b>";
            return;
        }

        $this->loadView( "web/datos", array(
                'data' => $res->items[0]
            ) );
    }

    private function apiRepository( $numdni )
    {
        $appCred = 'repos.vriunap:59d283';
        $pubKey  = 'ab01606f11c69eaf695e0877a2612696';

        $params = array(
            'dni'   => $numdni
        );

        $request = base64_encode( mcrypt_encrypt(MCRYPT_RIJNDAEL_256,
         $pubKey, json_encode($params), MCRYPT_MODE_ECB) );

        $api = curl_init();
        curl_setopt($api, CURLOPT_URL, "http://www.munilampa.gob.pe/apps/api");
        curl_setopt($api, CURLOPT_POST, TRUE );
        curl_setopt($api, CURLOPT_POSTFIELDS,
         array('request'=>$request,'id'=>$appCred));
        curl_setopt($api, CURLOPT_RETURNTRANSFER, 1);

        $resultado = curl_exec($api);

        return $resultado;
    }
}
?>
```

ANEXO B
CODIGO FUENTE DEL SERVIDOR

```
<?php defined(LX_APP) OR exit('No access allowed');

include("../absmain/mlLibrary.php");
include("../absmain/mlReniApi.php");


class Api extends CI_Controller {

    public function __construct()
    {
        parent::__construct();
        //$this->load->model("ado");
    }

    public function index()
    {
        header("Access-Control-Allow-Origin: *");
        header("Content-Type: application/json; charset=UTF-8");

        $acountuser = mlSecurePost("id");
        $argumentos = mlSecurePost("request");


                if( $acountuser == null ){
          $this->jsonResult( false, "Developers API 1.5" );
                        return;
                }

        $findUser = null;
                $arrUsers = array(
           "repos.munilamp:59d283" ,
                        "siga.munilampa:09c132" ,
         );

        // no existe usuario registrado
                if( !in_array( $acountuser, $arrUsers) ) {
           $this->jsonResult( false, "Developers API 1.5" );
                        return;
                }

        // buscamos que operacion desea
        $findUser = explode( ".", $acountuser );
        $findUser = $findUser[0];

        // el argumento de busqueda
```

```
    $args = $this->apiDecodArg( $argumentos );
            if( $args == null ) {
        $this->jsonResult( false, "Argument error" );
                    return;
            }

    //------------------------------------------------
    // 1. mostrar datos por DNI repositorio Perú.
    //------------------------------------------------
    if( $findUser == "repos" ){
        $data = $this->apiRepos( $args );
        $this->jsonResult( true, $data );
        return;
    }

    $this->jsonResult( false, "No results" );
}

private function apiRepos( $arg )
{
    if( ! isset($arg->dni) )
        return "null";

    //$res = reniGetArray( $arg->dni );
    $res = reniGetData( $arg->dni );

    // [ {"dni" : "0001"}, {"dni":"0002"} ]
    //
    $arrData = '['
            . '{'
            . ' "DNI" : "'.$res->DNI.'", '
            . ' "ApPaterno" : "'.$res->ApPaterno.'", '
            . ' "ApMaterno" : "'.$res->ApMaterno.'", '
            . ' "Nombres"   : "'.$res->Nombres.'" '
            . '}'
            . ']';

    return $arrData;
}

// convertir a una cadena de expresion json
private function jsonResult( $boolOK, $anyData )
{
    // = '{ "success" : false, "message" : "VRI Developer API" }';
    $arr = array( 'success' => false, 'message' => $anyData );

    // para evitar que le agregue las comillas "[]"
    //
```

```
    if( $boolOK )
        echo '{"success": true, "items":' .$anyData. '}';
    else
        echo json_encode( $arr );
}

private function apiDecodArg( $strb64 )
{
    // ecertis.vriunap:fbc1e3
    $pubKey  = 'ab01606f11c69eaf695e0877a2612696';

    $deco64 = base64_decode( $strb64 );
    $mdecry = mcrypt_decrypt( MCRYPT_RIJNDAEL_256, $pubKey, $deco64,
     MCRYPT_MODE_ECB);
    $mdecry = rtrim($mdecry, "\0");

    //
    // {"Codigo":"9908"} to stdClass Object
    //
    $data = json_decode($mdecry);

    // SECUENCIA CORRECTA:
    //--------------------------------------------------
    // Array ( [Codigo] => 9908 )           =>  $x = array()
    // {"Codigo":"9908"}                    =>  json_encode( $x )
    // stdClass Object ( [Codigo] => 9908 )  =>  json_decode( str $x )

    // en Objeto JSON
    return $data;
}

/*
//---------------------------------
public function prueba()
{
    $res = $this->apiGetCerti("01216522");
    print_r ( $res );

    $res = json_decode( $res );


    //echo "<hr>";
    //var_dump( $res );
    if( $res->success ) {
        //echo $res->items[0]->DNI;
        //echo $res->items[0]->ApPaterno;
    }
}
```

```
    private function apiGetCerti( $numdni )
    {
        $appCred = 'repos.vriunap:59d283';
        $pubKey  = 'ab01606f11c69eaf695e0877a2612696';

        $params = array(
            'dni'   => $numdni
        );

        $request = base64_encode( mcrypt_encrypt(MCRYPT_RIJNDAEL_256,
         $pubKey, json_encode($params), MCRYPT_MODE_ECB) );

        $api = curl_init();
        curl_setopt($api, CURLOPT_URL, "http://vriunap.pe/apps/api");
        curl_setopt($api, CURLOPT_POST, TRUE );
        curl_setopt($api, CURLOPT_POSTFIELDS, array( 'request'=>$request,
         'id'=>$appCred) );
        curl_setopt($api, CURLOPT_RETURNTRANSFER, 1);

        $resultado = curl_exec($api);

        //return $resultado;
        return $resultado;
    }
    */
}
```

ANEXO C
CODIGO FUENTE DE LA INTERFAZ WEB

```
<!DOCTYPE html>
<html lang="en">
 <head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-
      fit=no">
  <meta name="description" content="">
  <meta name="author" content="">
  <title> API REST 2017 </title>

  <!-- Bootstrap core CSS -->
  <link rel="stylesheet" href="http://www.atlasestateagents.co.uk/css/tether.min.css">
  <script src="http://www.atlasestateagents.co.uk/javascript/tether.min.js"></script>

  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0-
      alpha.6/css/bootstrap.min.css">
  <link rel="stylesheet" href="<?=base_url()?>includes/css/general.css">

  <script src="https://code.jquery.com/jquery-3.2.1.min.js"></script>
  <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0-
      alpha.6/js/bootstrap.min.js"></script>
  <script src="<?=base_url()?>includes/lightajax.js"></script>
  <script src="<?=base_url()?>includes/eventos.js"></script>
 </head>

 <body>

  <div class="collapse bg-inverse" id="navbarHeader">
   <div class="container">
    <div class="row">
     <div class="col-sm-8 py-4">
      <h4 class="text-white">Autores</h4>
       <p class="text-muted">
          Desarrollado como repositorio de datos para la <b>Municipalidad de
     Lampa</b>
          <br> &bullet; David Mamani Machaca
          <br> &bullet; Dilmerd Atencio Flores
          </li>
       </p>
     </div>
     <div class="col-sm-4 py-4">
      <h4 class="text-white">Contáctos</h4>
      <ul class="list-unstyled">
       <li><a href="" class="text-link">email: daf_2005@gmail.com</a></li>
       <li><a href="" class="text-link">email: dmm_2005@gmail.com</a></li>
```

```
        </ul>
       </div>
     </div>
   </div>
</div>
<div class="navbar navbar-inverse bg-inverse">
  <div class="container d-flex justify-content-between">
    <a href="#" class="navbar-brand"> MuniLampa (API REST) </a>
    <button class="navbar-toggler" type="button" data-toggle="collapse" data-
     target="#navbarHeader" aria-controls="navbarHeader" aria-expanded="false"
     aria-label="Toggle navigation">
      <span class="navbar-toggler-icon"></span>
    </button>
  </div>
</div>

<section class="jumbotron text-center" style="padding: 4rem; color: white; border-
     radius: 0px">
  <div class="container">
    <h1 class="jumbotron-heading">Repositorio</h1>
    <p class="lead">
       Datos completos de ciudadanos peruanos
    </p>
    <p>
      <a href="<?=base_url()?>" class="btn btn-primary"> Buscar Datos </a>
      <a href="<?=base_url("includes/arts/Manual.pdf")?>" class="btn btn-outline-
     success" target=_blank> Ver Documentación </a>
    </p>
  </div>
</section>

<div class="album text-muted">
  <div class="container">

    <div class="row">
       <div class="col-md-4" style="background: white"></div>

       <div class="col-md-4" style="background: white">
          <form method="post" name="frmx" id="frmx" onsubmit="return inBuscar()">
             <div class="form-group"> <br>
                <label for="dni">Ingrese número de DNI</label>
                <input type="number" class="form-control" name="dni" id="dni"
     placeholder="número de DNI" autofocus>
                <small class="form-text text-muted">este número será
     verificado</small>
             </div>
          </form>
       </div>
```

```
            <div class="col-md-4" style="background: white"></div>

            <div class="col-md-3" style="background: white"></div>
            <div class="col-md-6" style="background: white" id="resu">
            </div>
            <div class="col-md-3" style="background: white"></div>
          </div>

        </div>
      </div>

      <footer class="text-muted">
        <div class="container">
          <p class="float-right">
            <a href="#">Regresar a Inicio</a>
          </p>
          <p> &copy; Desarrollado para la Municipalidad de Lampa </p>
        </div>
      </footer>

    </body>
</html>
```

ANEXO D

CODIGO FUENTE DE LOS ESTILOS CSS

```
body {
  min-height: 30rem; /* Can be removed; just added for demo purposes */
}

.text-link{
  font-size: 0.7rem;
  font-color: orange;
  text-decoration: none;
}

.navbar {
  margin-bottom: 0;
}

.jumbotron {
  padding-top: 6rem;
  padding-bottom: 6rem;
  margin-bottom: 0;
  background: url("../img/blur.jpg");
  background-size: cover;
}
```

```
.jumbotron p:last-child {
  margin-bottom: 0;
}

.jumbotron-heading {
  font-weight: 300;
}

.jumbotron .container {
  max-width: 40rem;
}

.album {
  padding-top: 3rem;
  padding-bottom: 3rem;
  background-color: #f7f7f7;
}

.card {
  float: left;
  width: 33.333%;
  padding: .75rem;
  margin-bottom: 2rem;
  border: 0;
}

.card > img {
  margin-bottom: .75rem;
}

.card-text {
  font-size: 85%;
}

footer {
  padding-top: 3rem;
  padding-bottom: 3rem;
}

footer p {
  margin-bottom: .25rem;
}



/*
```

```
@import
        url('https://fonts.googleapis.com/css?family=Ubuntu:300,300i,400,400i,500,500i
        ,700,700i');

body{
    font-family:'Ubuntu', Arial;
    font-size: 160%;
    background: #FFF0C0;
}

.navbar-inverse {
    background-color: #e4a700;
    border-color: #080808;
}

.navbar-inverse .navbar-nav > .active > a, .navbar-inverse .navbar-nav > .active >
        a:hover, .navbar-inverse .navbar-nav > .active > a:focus {
    color: #ffffff;
    background-color: rgba(192, 141, 0, 0.71);
}

.panel-default {
    border-color: #999;
}

p{
    text-align: justify;
}

.clsFooter {
        font-size: 11px;
        font-family: verdana;
        background-color: #f0f0f0;
        border:  1px solid #D0D0D0;
        color:   #757575;
        padding: 12px 10px 12px 10px;
        text-align: center;
        margin-top: 7px;
}

h3{
    font-family: 'Verdana';
    font-size: 20px;
    color: #99BBCC;
    border-bottom: 3px solid #AADDEE;
    padding-bottom: 10px;
    margin-bottom: 25px;
}
```

*/

ANEXO E

CODIGO FUENTE DE MANIPULADOR JS LIGHTAJAX.JS Y EVENTOS.JS

```
/*******************************************************
 *  Area de funciones AJAX
 *
 *  Desarrolladores :
 *    Dilmerd Atencio Flores & David Mamani Machaca
 *
 *******************************************************/

function loadWeb( div, urlArgs )
{
   jVRI('#'+div).html( "Cargando..." );
   jVRI('#'+div).load( urlArgs );
}

function loadWebFrm( div, url, form )
{
   jVRI('#'+div).html( "Buscando..." );
   jVRI('#'+div).load( url, new FormData(form) );
}

function inBuscar()
{
   loadWebFrm( 'resu', 'inicio/inBuscar', frmx );
   return false;
}


// autoexecutables functions
//
(function ($) {
        // do something  $  is an argument
})("fuckQuery v.0.5.b");


//---------------------------------------------------------------------
// our DOM Request
//---------------------------------------------------------------------
function privSendRequest( qryFile, argObj, evOnDone, divDest )
{
   var ajax = new XMLHttpRequest();
```

```
    ajax.open( "POST", qryFile, true ); // async: no wait
    if( argObj != "[object FormData]" )
        ajax.setRequestHeader(
            "Content-Type",
            "application/x-www-form-urlencoded"
    );

    // stringURL or FormData
    ajax.send( argObj );
    ajax.onreadystatechange = function() {
        if ( ajax.readyState == 4 ) {
            if( evOnDone != null )
                evOnDone( ajax.responseText );
            if( divDest != null )
                divDest.innerHTML = ajax.responseText;
        }
    }
}

//-----------------------------------------------------------
//
//-----------------------------------------------------------
function mlElementById( strid )
{
    return document.getElementById( strid );
}

//-----------------------------------------------------------
function jsonStr( vars ) {
    // in  = { lastname:'Jose', firstname:'Xavier' }
    // out = lastname=jose...
    arr = JSON.stringify( vars );
    str = arr.split(":").toString();
    str = str.replace( "{", "(" );
    str = str.replace( "}", ")");
    arr = eval( 'Array' + str );
    tmp = "";
    for( i=0; i<arr.length; i+=2 ){
        if( tmp ) tmp += "&";
        tmp += arr[i+0] +"="+ arr[i+1];
    }

    return tmp;
}
//-----------------------------------------------------------
function mlObjectDOM( ctrl ) {

    this.control = ctrl;
```

```
    //-------------------------------------------
    this.val = function ( arg ) {

        if( arg!=null ) this.control.value = arg;
        else {
            if( this.control == null )
                return 0;

            return this.control.value;
        }
    };

    this.html = function( arg ) {

        this.control.innerHTML = arg;
    };

    this.focus = function(){

        this.control.focus();
    };

    this.change = function( argfunc ){

        this.control.addEventListener('change', argfunc, false);
    };

    this.show = function(){
        this.control.style = "display: block !important;";
    }

    this.load = function( url, arg ) {

        // arguments r: url, arg, no-event, div
        privSendRequest( url, arg, null, this.control );
    };


    //this.disabled = function( arg ){
    //    this.control.disabled = arg? true : false;
    //}

    this.style = function( arg ) {
        ;
    };
};
//-----------------------------------------------------------
var mlLightAjax = function( strId ) {
```

```
        // this.params = params; {a:0}
        //
        // intead of: document.getElementById(idStr)

    return ( new mlObjectDOM(document.querySelector(strId)) );
};


mlLightAjax.ajax = function( params ) {

    privSendRequest(
            params.url,
            //params.type,  ::no used
            params.data,
            params.success,
            null   // no-div
    );
};

//------------------------------------------------------
// local definition over our Compatible fuckQuery
//------------------------------------------------------
//$ = mlLightAjax;


jVRI = mlLightAjax;
//------------------------------------------------------

/*
    $(document).ready(function() {
    $("#word_count").on('keyup', function() {
        var words = this.value.match(/\S+/g).length;
        if (words > 10) {
            var trimmed = $(this).val().split(/\s+/, 10).join(" ");
            $(this).val(trimmed + " ");
        }
        else {
            $('#display_count').text(words);
            $('#word_left').text(10-words);
        }
    });
 });
*/
```