Proyecto de Análisis, Diseño y Desarrollo

Implementación de API REST

19/11/2015
Utilizando servicios a través de una API REST
Rudy Donaldo Marroquín García
1490-09-9429

Ing. Iván Antonio de León Fuentes
Evaluador
Evaluación Privada

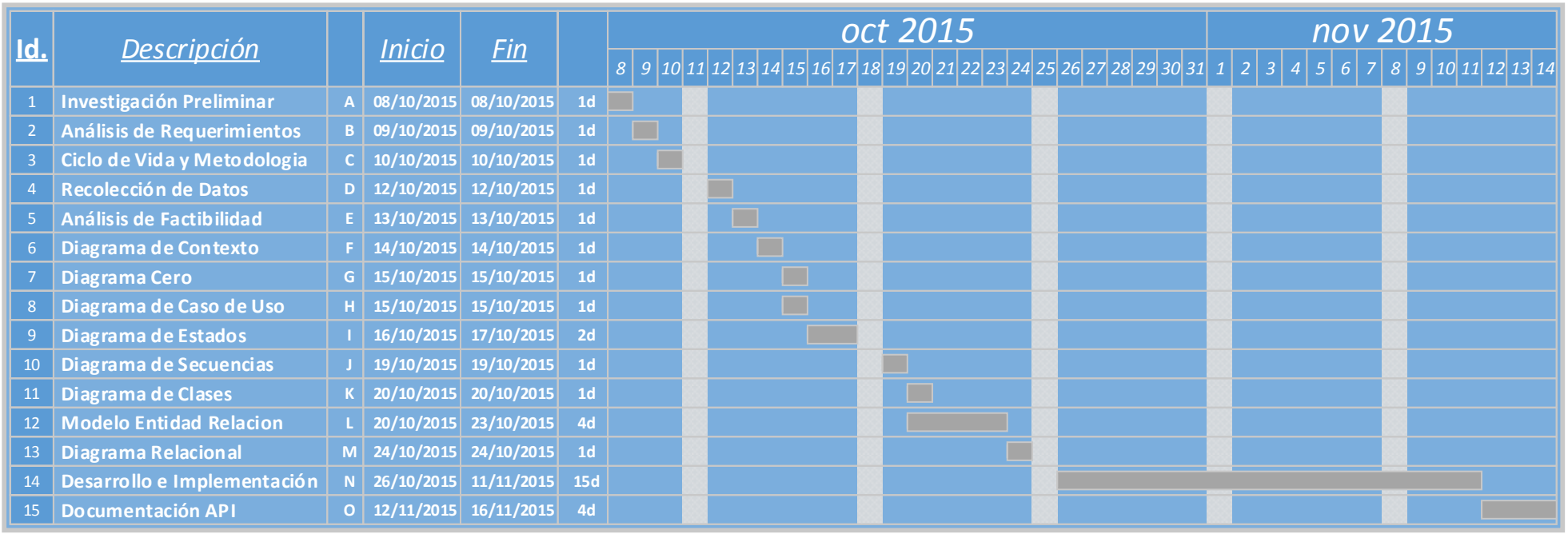
INDICE

Programa de Actividades

	Actividades	Predecesor	Duración	Unidad	Encargado
A	Investigación Preliminar	---	1	día	Rudy Marroquín
B	Análisis de Requerimientos	A	1	día	Rudy Marroquín
C	Ciclo de Vida y Metodología	B	1	día	Rudy Marroquín
D	Recolección de Datos	C	1	día	Rudy Marroquín
E	Análisis de Factibilidad	D	1	día	Rudy Marroquín
F	Diagrama de Contexto	E	1	día	Rudy Marroquín
G	Diagrama Cero	F	1	día	Rudy Marroquín
H	Diagrama de Caso de Uso	F	1	día	Rudy Marroquín
I	Diagrama de Estados	H	2	día	Rudy Marroquín
J	Diagrama de Secuencia	I	1	día	Rudy Marroquín
K	Diagrama de Clases	J	1	día	Rudy Marroquín
L	Modelo Entidad Relación	J	4	día	Rudy Marroquín
M	Diagrama Relacional	L	1	día	Rudy Marroquín
N	Desarrollo e Implementación	M	15	día	Rudy Marroquín
O	Documentación API	N	4	día	Rudy Marroquín

Diagrama de Actividades (GANTT)

Id.	Descripción		Inicio	Fin	
1	Investigación Preliminar	A	08/10/2015	08/10/2015	1d
2	Análisis de Requerimientos	B	09/10/2015	09/10/2015	1d
3	Ciclo de Vida y Metodologia	C	10/10/2015	10/10/2015	1d
4	Recolección de Datos	D	12/10/2015	12/10/2015	1d
5	Análisis de Factibilidad	E	13/10/2015	13/10/2015	1d
6	Diagrama de Contexto	F	14/10/2015	14/10/2015	1d
7	Diagrama Cero	G	15/10/2015	15/10/2015	1d
8	Diagrama de Caso de Uso	H	15/10/2015	15/10/2015	1d
9	Diagrama de Estados	I	16/10/2015	17/10/2015	2d
10	Diagrama de Secuencias	J	19/10/2015	19/10/2015	1d
11	Diagrama de Clases	K	20/10/2015	20/10/2015	1d
12	Modelo Entidad Relacion	L	20/10/2015	23/10/2015	4d
13	Diagrama Relacional	M	24/10/2015	24/10/2015	1d
14	Desarrollo e Implementación	N	26/10/2015	11/11/2015	15d
15	Documentación API	O	12/11/2015	16/11/2015	4d



INVESTIGACIÓN PRELIMINAR

ANÁLISIS DEL PROBLEMA Y OPORTUNIDADES

Clientes:

Página Web, Aplicación Móvil y Aplicación de Escritorio

Descripción del Problema:

Actualmente el desarrollo de sistemas de bases de datos se ha enfocado en el desarrollo web y en el desarrollo móvil, ya que la mayoría de personas tienen fácil acceso a internet.

La seguridad es lo más importante para que los usuarios tengan confianza de realizar transacciones básicas y que no corra riesgo su información.

Para lo cual se necesita una herramienta flexible en cualquier entorno véase dispositivo móvil, web o aplicación de escritorio la cual nos permita poder acceder a la información no importando la plataforma que estemos trabajando.

Posible Solución:

Se propone la implementación un WEB SERVICE REST dado que es una de la mejores opciones en este momento hablando de servicios web es la tendencia actual.

No es un método nuevo o algo novedoso este existe desde hace mucho tiempo, solo que se ha visto su gran utilidad o potencial por el aumento de nuevas tecnologías en el desarrollo de múltiple plataformas o dispositivos, a los cuales debe llegar la información de forma consistente y segura, por los cual utilizar esta herramienta es una solución apropiada.

Su fácil utilización para los usuarios de cualquier tecnología es muy sencilla y fácil de implementar, porque esta está basada en peticiones las cuales se generan de forma sencilla, e incorporan seguridad a través de usuarios y contraseñas llamadas tokens, hay que tener en cuenta que este sistema solo atiende peticiones de los usuarios registrados o que tienen acceso a la aplicación.

Las consultas o peticiones hechas por los usuarios no son recordadas por el servicio REST, por lo cual se recalca que por cada petición se debe enviar el tokens de autenticación para poder procesar la petición, esto permite que el servicio REST siempre esté disponible y sus respuestas sean más rápidas no importando el incremento de peticiones por los usuarios, por lo que en la actualidad este servicio está cambiando la forma de servicios web.

OBJETIVOS:

Generales:

- Crear un medio de comunicación cómodo y fácil de utilizar permitiendo a sus usuarios una experiencia agradable.

- Ser una herramienta que se acomoda a las necesidades del usuario no importando en la tecnología que está desarrollado.

- Tener una herramienta de información disponible en cualquier momento y en cualquier dispositivo solo con tener acceso a internet.

Específicos

- Transmitir información de forma segura y confiable.

- Implementar nuevas tecnologías las cuales nos permiten ser flexibles en cualquier tecnología.

- Tener un sistema escalable y consistente en la gestión de peticiones.

- Desarrollar sistemas de calidad.

ANÁLISIS DE REQUERIMIENTOS

DETERMINACIÓN DE REQUERIMIENTOS

¿Qué es lo que quiere el cliente?

Contar con un sistema de gestión de almacenamiento y transmisión de información, en el cual puedan manejarse dicha información de forma fácil y agradable, los usuarios podrán editar la información disponible en este caso como Publicaciones, comentarios, categorías, usuarios y tipos de usuarios.

Se requiere que para estas entidades se puedan hacer estas operaciones Selección, inserción, actualización y eliminación (CRUD) para la gestión de información.

ESPECIFICACIÓN DE REQUERIMIENTOS

¿Cuáles son los procesos básicos?

El proceso básico o las transacciones son Seleccionar, Insertar, Actualizar y Eliminar ya sea usuarios, comentarios, publicaciones, categorías, tipos de usuarios, estos pueden ser filtrados por su identificador (id) para gestionar una opción especifica cómo puede ser actualizar una publicación, eliminar una categoría o insertar un usuario, etc.

¿Qué datos utiliza o produce este proceso?

ESPIRAL

El modelo de la espiral es un modelo orientado a riesgo que divide el proyecto de software en mini proyectos. Cada proyecto se encargará de resolver uno o varios riesgos hasta que estén todos controlados. Una vez que estén los riesgos más importantes controlados se finaliza igual que el ciclo de vida en cascada. En el ciclo de vida en espiral localizan los riesgos, genera un plan para manejarlos y se establece una aproximación a la siguiente iteración. Con cada iteración se produce una aproximación al producto final.

En el modelo en espiral se comienza con una parte pequeña del proyecto y se expande tras reducir los riesgos para la siguiente iteración.
En cada iteración seguimos los siguientes pasos:

- ✓ Determinar objetivos, alternativas y límites.
- ✓ Identificar y resolver riesgos.
- ✓ Evaluar las alternativas.
- ✓ Generar entregas de esta iteración, y comprobar que son correctas.
- ✓ Planificar la siguiente iteración.

Si se decide ejecutar la siguiente iteración, hay que establecer un enfoque para ella.
En este modelo las primeras iteraciones son menos costosas y a medida que se avanza aumenta el costo.

Las ventajas de este modelo son:

- ✓ Se disminuyen los riesgos.
- ✓ Al final de cada iteración se obtienen los puntos de verificación.
- ✓ Se obtienen con anterioridad indicaciones de cualquier riesgo insuperable.

Las desventajas de este modelo son:

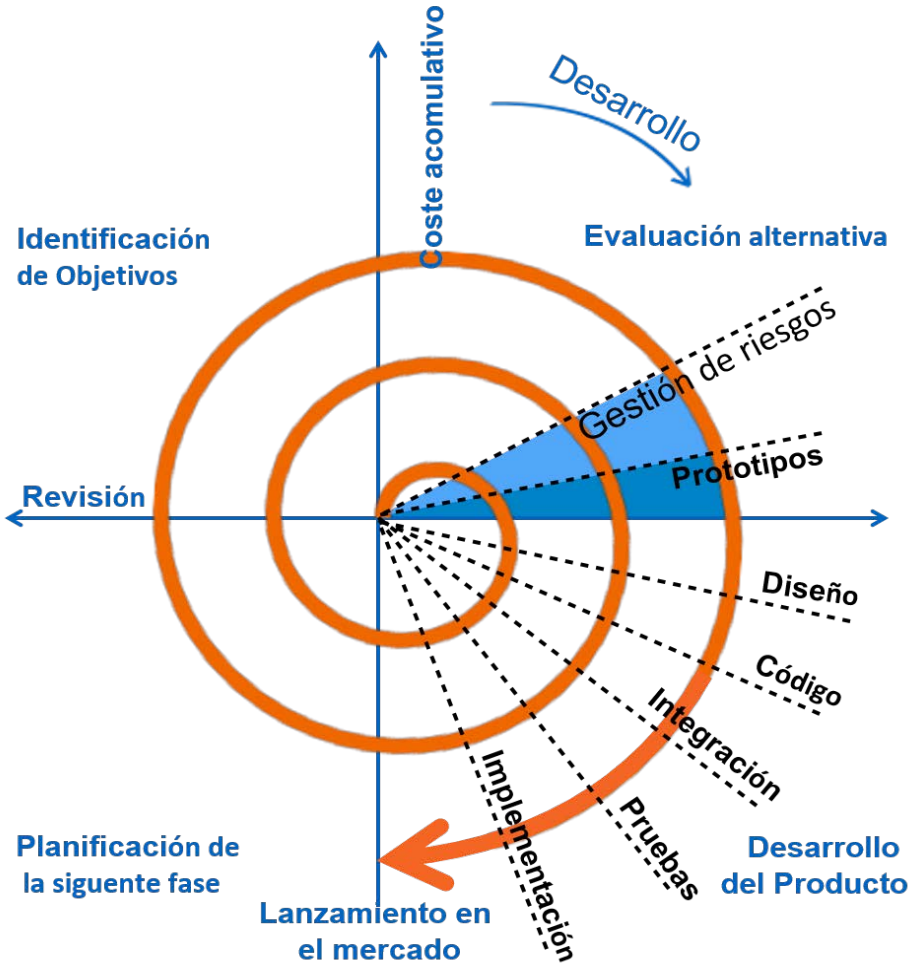
- ✓ Un aumento de costos.
- ✓ Es un modelo complicado de llevar a cabo porque exige una gestión en la cual siempre se está monitoreando, y unos conocimientos cada vez más profundos.

El análisis del riesgo se hace de forma explícita y clara. Une los mejores elementos de los restantes modelos.

- ✓ Reduce riesgos del proyecto
- ✓ Incorpora objetivos de calidad
- ✓ Integra el desarrollo con el mantenimiento, etc.

Además es posible tener en cuenta mejoras y nuevos requerimientos sin romper con la metodología, ya que este ciclo de vida no es rígido ni estático.
Para cada ciclo habrá cuatro actividades:

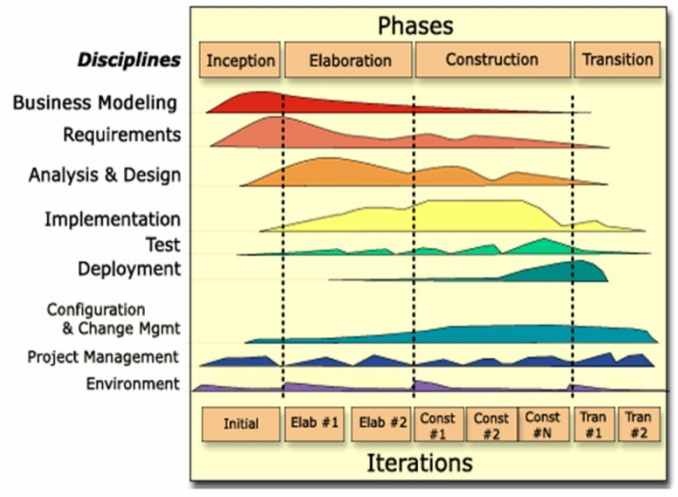
1. Determinar Objetivos.
2. Análisis del riesgo.
3. Desarrollar y probar.
4. 'Planificación.'



MODELO DE DESARROLLO

RUP

Es un proceso de ingeniería de software, que hace una propuesta orientada por disciplinas para lograr las tareas y responsabilidades de una organización que desarrolla software.
Su meta principal es asegurar la producción de software de alta calidad que cumpla con las necesidades de los usuarios, con una planeación y presupuesto predecible.



- Horizontalmente se representa el tiempo y muestra los aspectos del ciclo de vida del proceso. Representa el aspecto dinámico del proceso a través de las fases, iteraciones y productos intermedios.
- Verticalmente representa las disciplinas que agrupan actividades por su naturaleza. Representa el aspecto estático del proceso a través de componentes, disciplinas, actividades, flujos de trabajo, artefactos y roles.

En cuanto a tiempo el ciclo de vida de RUP se descompone en 4 FASES secuenciales, cada cual concluye con un producto intermedio. Al terminar cada fase se realiza una evaluación para determinar si se ha cumplido o no con los objetivos de la misma.
Las fases son:

- ✓ Inicio
- ✓ Elaboración
- ✓ Construcción
- ✓ Transición.

FASES.

1. Inicio

El objetivo general de esta fase es establecer un acuerdo entre todos los interesados acerca de los objetivos del proyecto. Esta fase es significativamente primaria para el desarrollo de nuevo software, ya que se asegura de identificar los riesgos relacionados con el negocio y requerimientos. Para proyectos de mejora de software existente esta fase es más breve y se centra en asegurar que vale la pena y es posible, desarrollar el proyecto.

2. Elaboración

El objetivo en esta fase es establecer la arquitectura base del sistema para proveer bases estables para el esfuerzo de diseño e implementación en la siguiente fase. La arquitectura debe abarcar todas las consideraciones de mayor importancia de los requerimientos y una evaluación del riesgo.

3. Construcción

El objetivo de la fase de construcción es clarificar los requerimientos faltantes y completar el desarrollo del sistema basados en la arquitectura base. Vista de cierta forma esta fase es un proceso de manufactura, en el cual el énfasis se torna hacia la administración de recursos y control de las operaciones para optimizar costos, tiempo y calidad.

4. Transición

Esta fase se enfoca en asegurar que el software esté disponible para sus usuarios. Esta fase se puede subdividir en varias iteraciones, además incluye pruebas del producto para poder hacer el entregable del mismo, así como realizar ajuste menores de acuerdo a ajuste menores propuestos por el usuario. En este punto, la retroalimentación de los usuarios se centra en depurar el producto, configuraciones, instalación y aspectos sobre utilización.

Disciplinas

Una disciplina es una colección de actividades relacionadas con un área de atención dentro de todo el proyecto. El grupo de actividades que se encuentran dentro de una disciplina principalmente son una ayuda para entender el proyecto desde la perspectiva clásica de cascada.
Las disciplinas son:

- ✓ Modelado de Negocios,
- ✓ Requerimientos,
- ✓ Análisis y Diseño,
- ✓ Implementación,
- ✓ Pruebas, Transición,
- ✓ Configuración y Administración del Cambio,
- ✓ Administración de Proyectos y Ambiente.

Modelado de Negocios

Los propósitos que tiene el Modelo de Negocios son:

- Entender los problemas que la organización desea solucionar e identificar mejoras potenciales.
- Medir el impacto del cambio organizacional.
- Asegurar que clientes, usuarios finales, desarrolladores y los otros participantes tengan un entendimiento compartido del problema.
- Derivar los requerimientos del sistema de software, necesarios para dar soporte a los objetivos de la organización.
- Entender como el sistema a ser desarrollado entra dentro de la organización.

Requerimientos

Esta disciplina tiene el propósito de:

- Establecer y mantener un acuerdo con los clientes y los otros interesados acerca de que debe hacer el sistema.
- Proveer a los desarrolladores del sistema de un mejor entendimiento de los requerimientos del sistema.
- Definir los límites (o delimitar) del sistema.
- Proveer una base para la planeación de los contenidos técnicos de las iteraciones.
- Proveer una base para la estimación de costo y tiempo necesarios para desarrollar el sistema.
- Definir una interfaz de usuario para el sistema, enfocada en las necesidades y objetivos del usuario.

Análisis y Diseño

El propósito del Análisis y Diseño es:

- Transformar los requerimientos a diseños del sistema.
- Desarrollar una arquitectura robusta para el sistema.
- Adaptar el diseño para hacerlo corresponder con el ambiente de implementación y ajustarla para un desempeño esperado.

Implementación

El propósito de la implementación es:

- Definir la organización del código, en términos de la implementación de los subsistemas organizados en capas.
- Implementar el diseño de elementos en términos de los elementos (archivos fuente, binarios, ejecutables y otros)
- Probar los componentes desarrollados como unidades.
- Integrar los resultados de los implementadores individuales en un sistema ejecutable.

La disciplina de implementación limita su alcance a como las clases individuales serán probadas. Las pruebas del sistema son descritas en futuras disciplinas.

Pruebas

Esta disciplina actúa como un proveedor de servicios a las otras disciplinas en muchos aspectos. Pruebas se enfoca principalmente en la evaluación y aseguramiento de la calidad del producto.

Transición

- Esta disciplina describe las actividades asociadas con el aseguramiento de la entrega y disponibilidad del producto de software hacia el usuario final.
- Existe un énfasis en probar el software en el sitio de desarrollo, realización de pruebas beta del sistema antes de su entrega final al cliente.

Administración y Configuración del Cambio

Consiste en controlar los cambios y mantener la integridad de los productos que incluye el proyecto.

Los métodos, procesos y herramientas usadas para proveer la administración y configuración del cambio pueden ser considerados como el sistema de administración de la configuración.

Administración de Proyectos

El propósito de la Administración de Proyectos es:

- Proveer un marco de trabajo para administrar los proyectos intensivos de software.
- Proveer guías prácticas para la planeación, soporte, ejecución y monitoreo de proyectos.
- Proveer un marco de trabajo para la administración del riesgo.
  - Ambiente
- Se enfoca en las actividades necesarias para configurar el proceso al proyecto.
- Describe las actividades requeridas para desarrollar las líneas guías de apoyo al proyecto.
- El propósito de las actividades de ambiente es proveer a las organizaciones de desarrollo de software del ambiente necesario (herramientas y procesos) que den soporte al equipo de desarrollo.

MARCO TEORICO

INTERNET

Es una comunicación contante entre servidor y clientes hacer que esa comunicación sea lo más eficaz posible es el principal motivo de todas las tecnologías Web, desde hace unos años los servicios web no han parado de aumentar y mejorar.

INTERNET y su Evolución

Inicialmente Internet tenía un objetivo claro. Se navegaba en Internet para algo muy concreto: búsquedas de información, generalmente. Ahora quizás también, pero sin duda algún hoy es más probable perderse en la red, debido al inmenso abanico de posibilidades que brinda. Hoy en día, la sensación que produce Internet es un ruido, una serie de interferencias, una explosión de ideas distintas, de personas diferentes, de pensamientos distintos de tantas posibilidades que, en ocasiones, puede resultar excesivo. El crecimiento o, más bien, la incorporación de tantas personas a la red hacen que las calles de lo que en principio era una pequeña ciudad llamada Internet se conviertan en todo un planeta extremadamente conectado entre sí, entre todos sus miembros. El hecho de que Internet haya aumentado tanto implica una mayor cantidad de relaciones virtuales entre personas. Es posible concluir que cuando una persona tenga una necesidad de conocimiento no escrito en libros, puede recurrir a una fuente más acorde a su necesidad, ahora esta fuente es posible en Internet. Como toda gran revolución, Internet augura una nueva era de diferentes métodos de resolución de problemas creados a partir de soluciones anteriores. Internet produce algo que todos han sentido alguna vez; produce la esperanza que es necesaria cuando se quiere conseguir algo. Es un despertar de intenciones que jamás antes la tecnología había logrado en la población mundial.

Aplicación del lado del Servidor

Una aplicación del lado del servidor es cualquier programa o conjunto de instrucciones diseñadas con la finalidad de que un Servidor Web las procese para realizar alguna acción. Las aplicaciones del lado del servidor están escritas mediante algún lenguaje de programación, entre los que destacan:

Lenguaje	Fecha de primera versión estable
PHP	1995
ASP	1998
Perl	1987
Python	1991
Ruby	1995

El 75% de las aplicaciones del lado del servidor están escritas en PHP, seguido de ASP y las demás opciones usadas de forma alternativa y muy casual.

Métodos HTTP

Una API RESTful bien diseñada debe soportar la mayoría de métodos HTTP comúnmente utilizados (GET, POST, PUT y DELETE). Existen otros métodos HTTP tales como OPTIONS, HEAD pero no son utilizados frecuentemente. Cada método debe ser utilizado dependiendo al tipo de operación que se desea realizar.

GET	**Para obtener un recurso**
POST	**Para crear un nuevo recurso**
PUT	**Para actualizar un recurso existente**
DELETE	**Para eliminar un recurso**

Las peticiones al servidor suelen realizarse mediante HTTP utilizando el método de petición GET, en el que el recurso se solicita a través de la url al servidor Web.

```
GET /index.html HTTP/1.1 HOST: www.host.com
```

En la barra de URL de un navegador cualquiera, la petición anterior sería análoga a la siguiente dirección Web:

```
www.host.com/index.html
```

Nombre	Descripción
Hipervínculo enlace **o** link	Es una porción de contenido Web, texto, imagen y otros elementos, que enlaza con una dirección Web. Al pulsar un hipervínculo, el navegador genera una petición GET automática a la dirección URL de dicho link.

Formulario Web	Al realizar el envío satisfactorio de los datos de un formulario, el navegador Web genera una petición GET o POST (comúnmente POST) automática a la par que envía los datos al servidor.
Barra de direcciones	Todos los navegadores incluyen una barra de direcciones mediante la cual puede accederse manualmente a cualquier dirección URL, de modo que el navegador generará una petición GET automática a dicha URL cada vez que el usuario lo desee.
Script activo **o** pasivo	Cualquier aplicación Javascript tiene acceso al estado del navegador, cómo puede modificar los datos que describen tal estado, de forma pasiva (sin medio de la intervención del usuario) o de forma activa (mediante alguna acción del usuario).

Código de Estados de HTTP

Los códigos de estado HTTP en el cuerpo de respuesta informa a la aplicación del cliente qué acción debe realizarse con esa respuesta. Por ejemplo si la respuesta fuese 200, significa que en el lado del servidor la consulta se procesó de manera satisfactoria. Si fuese 401, significa que la petición se hizo sin autorización. Un ejemplo típico de ello es que la api key no es válida.

No es necesario dar soporte a todos los códigos HTTP, pero al menos los siguientes

200	**OK**
201	**Creado**
304	**No modificado**
400	**Petición incorrecta**
401	**No autorizado**
403	**Prohibido**
404	**No encontrado**
422	**Entidad imposible de procesar**
500	**Error interno del servidor**

Un servidor web o servidor HTTP es un programa informático que procesa una aplicación del lado del servidor, realizando conexiones bidireccionales y/o unidireccionales y síncronas o asíncronas con el cliente y generando o cediendo una respuesta en cualquier lenguaje o Aplicación del lado del cliente. El código recibido por el cliente suele ser

compilado y ejecutado por un navegador web. Para la transmisión de todos estos datos suele utilizarse algún protocolo. Generalmente se usa el protocolo HTTP para estas comunicaciones, perteneciente a la capa de aplicación del modelo OSI. El término también se emplea para referirse al ordenador que ejecuta el programa.

Estructura de la URL

En el diseño del REST las direcciones URL deben estar bien formadas y fáciles de entender. Cada URL para un recurso debería ser único. Si tu API requiere una clave (API Key) para acceder, la api key debe estar incluida en el encabezado HTTP en lugar de incluirlo en la URL.

Por ejemplo:

GET http://gifeditor.net/api/v1/gifs/11

- Retornará los detalles de un gif cuyo id es 11

POST http://gifeditor.net/api/v1/gifs

- Subirá un nuevo gif

REST

Es un producto del siglo XX, es un estilo de arquitectura de software muy vinculado desde sus orígenes al protocolo HTTP que consiste en una serie de directrices y mejores prácticas para mejorar las comunicaciones de Cliente-Servidor y ofrecer servicios web escalables.

Principios que Definen una Arquitectura REST

1. **Todo es un Recurso:** se parte de la idea que todos los datos se representan en un formato específico que tienen y no por un archivo físico, Cada trozo de dato en internet tiene un formato que se describe por el tipo de contenido ejemplo.

JPEG	Imagen
MPEG	Video
HTML	texto
XML	texto

2. Cada recurso tiene que ser identificable por un identificador único URI en internet hay muchos tipos de recurso y cada uno debe ser diferente a todos los demás por lo cual debe de existir una URI única que los identifique de los demás.

3. Usar los métodos Estándar HTTP, en los cuales se definen ocho acciones distintas.

GET	POST	PUT	DELETE	HEAD	OPTIONS	TRACE	CONNECT

4. Los recursos tienen múltiples representaciones, una característica clave es que los recursos pueden tener distintos formatos que son diferentes a en los que están almacenados por ejemplo un recurso XML puede tener una representación JSON.

5. Comunicaciones sin estado, no se pueden mantener ningún tipo de persistencia en las transacciones, el servidor trata cada transacción como algo total mente independiente y la comunicación siempre devuelve una estado final.

API

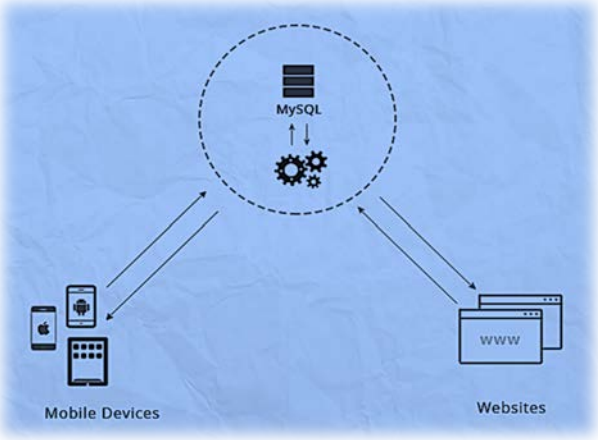
En la programación de computadoras, una interfaz de programación de aplicaciones (API) es un conjunto de rutinas, protocolos y herramientas para la construcción de aplicaciones de software. Una API expresa un componente de software en términos de sus operaciones, entradas, salidas, y tipos subyacentes. Una API define funcionalidades que son independientes de sus respectivas implementaciones, lo que permite definiciones e implementaciones para variar sin comprometer la interfaz. Una buena API hace que sea más fácil desarrollar un programa, proporcionando todos los bloques de construcción. Los programadores a continuación, poner los bloques juntos.

Arquitectura de API REST

Fundamentos del diseño de una API REST

La arquitectura REST es muy útil para construir un cliente/servidor para aplicaciones en red. REST significa (Representational State Transfer) de sus siglas en inglés. Implementar REST es muy sencillo comparado a otros métodos tales como SOAP, CORBA, WSDL básicamente funciona en el protocolo HTTP. Servicio REST no tiene estado (es stateless), lo que quiere decir que, entre dos llamadas, el servicio pierde todos sus datos. Esto quiere decir que el servidor no guarda información de las transacciones u operaciones que realiza cada transacción es diferente y separada aunque las peticiones sean parecidas él no las compara sencilla mente atiende cada petición por separado por ejemplo cuando ingresamos un usuario y una contraseña y esperamos que “nos recuerde” en cada petición que realicemos. De ahí el nombre: el estado lo mantiene

el cliente y por lo tanto es el cliente quien debe pasar el estado en cada llamada. Si quiero que un servicio REST me recuerde, debo pasarle quien soy en cada petición. Eso puede ser un usuario y una contraseña, un tokens o cualquier otro tipo de credenciales, pero debo pasarlas en cada petición. Y será lo mismo en cada petición no importando de qué tipo sea.



REST Y RESTfull

Cuando hablamos de REST y RESTfull no hablamos de temimos diferentes es más diferencias de matices. REST es un estilo de arquitectura de software que sigue los principios que acabamos de ver. RESTfull son los servicios web que siguen esos principios que implemente REST.

Ventajas de la utilización de REST

1. **Separación del recurso y la representación:** es un conjunto de información que puede tener múltiples representaciones y no tienen estados.
2. **Visibilidad:** está diseñado para ser simple, lo que significa que cada aspecto del servicio debe ser auto descriptivo siguiendo el lenguaje natural http, esto garantiza la comunicación clara de todas las comunicaciones http entre el servicio REST y el cliente que lo llama.
3. **Seguridad o Fiabilidad:** se garantiza que los métodos http son seguros lo que garantiza que no sufrirá ningún cambio en su estado al solicitar un recurso.
4. **Escalabilidad y Rendimiento:** en internet contiene una gran cantidad de datos y así mismo de usuarios he aquí la necesidad de la escalabilidad dependiendo de la cantidad de carga que se maneje, si el aumento de la demanda exige aumento del número de servicios, esto puede hacerse sin preocuparse por la sincronización entre los mismos, al no tener que preocuparse por el estado.

SOAP y REST

SOAP es el acrónimo de "Simple Object Access Protocol" y es el protocolo que se oculta tras la tecnología que comúnmente denominamos "Web Services" o servicios web. SOAP es un protocolo extraordinariamente complejo pensado para dar soluciones a casi cualquier necesidad en lo que a comunicaciones se refiere, incluyendo aspectos avanzados de seguridad, transaccionalidad, mensajería asegurada y demás. Cuando salió SOAP se vivió una época dorada de los servicios web. Aunque las primeras implementaciones eran lo que se llamaban WS1.0 y no soportaban casi ningún escenario avanzado, todo el mundo pagaba el precio de usar SOAP, ya que parecía claro que era el estándar que dominaría el futuro. Con el tiempo salieron las especificaciones WS-* que daban soluciones avanzadas, pero a la vez que crecían las capacidades de SOAP, crecía su complejidad. Al final, los servicios web SOAP terminan siendo un monstruo con muchas capacidades pero que en la mayoría de los casos no necesitamos.

Por su parte REST es simple. REST no quiere dar soluciones para todo y por lo tanto no pagamos con una demasiada complejidad una potencia que quizá no vamos a necesitar.

Interfaz Uniforme *(Uniform interface)*

Una diferencia fundamental entre un servicio web clásico (SOAP) y un servicio REST es que el primero está orientado a RPC, es decir, a invocar métodos sobre un servicio remoto, mientras que el segundo está orientado a recursos. Es decir, operamos sobre recursos, no sobre servicios. En una API REST la idea de "servicio" como tal desaparece. Lo que tenemos son recursos, accesibles por identificadores (URIs). Sobre esos recursos podemos realizar acciones, generalmente diferenciadas a través de verbos HTTP distintos. Así, en un servicio web clásico (SOAP) tendríamos un servicio llamado BeerService que tendría un método llamado GetAll que me devolvería todas las cervezas. La idea, independientemente de la tecnología usada para consumir el servicio web, es que se llama a un método (GetAll) de un servicio remoto (BeerService). Del mismo modo para obtener una cerveza en concreto llamaríamos al método GetById() pasándole el id de la cerveza. De ahí que se diga que están orientados a RPC (Remote Procedure Call – Llamada a método remoto).

Por su parte en un servicio REST la propia idea de servicio se desvanece. En su lugar nos queda la idea de un "recurso", llamémosle "Colección de cervezas" que tiene una URI que lo identifica, p. ej. /Beers. Así, si invoco

dicha URI debo obtener una representación de dicho recurso, es decir, debo obtener el listado de todas las cervezas.

Para obtener datos de una cerveza, habrá otro recurso (cerveza) con una URI asociada. Además, entre los recursos hay relaciones: está claro que una cerveza forma parte de la "Colección de cervezas" así que parece lógico que a partir de la colección entera (/Beers) pueda acceder a uno de sus elementos con una URI tipo /Beers/123, siendo "123" el ID de la cerveza. Volvamos a nuestro servicio SOAP: para dar de alta una cerveza llamaríamos a un método "AddBeer" de nuestro "BeerService", y le pasaríamos la cerveza a añadir. En cambio, en nuestra API REST añadir una cerveza consiste en usar la URI de dicha cerveza, (p. ej. /Beers/456 si estamos añadiendo la cerveza con ID 456) usando POST o PUT como verbo HTTP y colocando los datos de la cerveza en el cuerpo de la petición. La URI para acceder a la cerveza con ID 456 es siempre /Beers/456 y es el verbo HTTP (GET, POST, PUT, DELETE,...) el que indica cual es la operación que deseamos hacer con esa cerveza.

Versionado de las API

Mantener versiones de API ya sea para incluirlos en la URL o en el encabezado HTTP, aunque muchos prefieren utilizarlo en la URL especialmente porque resulta más sencillo migrar a una nueva versión.

Ejemplo:

http://gifeditor.net/api/v1/gifs

http://gifeditor.net/api/v2/gifs

Tipo de contenido

El tipo de contenido o Content Type en los encabezados HTTP especifica el tipo de dato que se debe trasmitir entre el servidor y el cliente. Dependiendo del tipo de dato a utilizar, necesitamos especificarlo.

Por ejemplo, JSON es un tipo MIME debe especificarse como Content-Type: application/json, para XML Content-Type: application/xml.

API Key

Una API Key no es más que un identificador (una clave y contraseña para autenticarte cada vez que utilizas el Webservices), y al cual se le asignarán todos los cargos que correspondan por uso de los distintos servicios.

(API key) es un código aprobado en los programas informáticos que llaman un (API de interfaz de programación de aplicaciones) para identificar el programa que llama, su desarrollador, o de su usuario al sitio Web. Claves de la API se utilizan para rastrear y controlar cómo se está utilizando el API, por ejemplo para prevenir el uso malicioso o abuso de la API (según se define tal vez por los términos de servicio).

La clave API a menudo actúa como un identificador único y un contador secreto para la autenticación, y por lo general tendrá un conjunto de derechos de acceso en el API asociado con él.

Claves de la API se pueden basar en el UUID del sistema para asegurar que será única para cada usuario.

La clave de la API debe ser incluida en cada petición en el encabezado en lugar de utilizar la URL.

Autorización: cfb7acfbb9bb8...

ANÁLISIS DE FACTIBILIDAD

TÉCNICA (CONOCIMIENTO)

- Características en hardware y software suficientes para implementar el sistema teniendo en cuenta que este se implemente en la nube.
- Acceso a internet.

OPERACIONAL (SUFICIENTE PERSONAL)

- **¿Existe demanda suficiente para el proyecto?**

  Este proyecto está respaldado por el consumo de esta nueva tecnología por lo cual se puede decir que su implementación y su utilización será comprobable.

- **Los servicios prestados por los WEB SERVISES REST, ¿son aceptados por los usuarios?**

  Son aceptados, y muchas aplicaciones web están optando migrar a utilizar un servicio REST por comodidad y escalabilidad.

- **¿Los usuarios han participado en la planeación y desarrollo del proyecto?, ¿Cómo lo han hecho?**

  Si ellos dieron los lineamientos de como quisieran que funcionara el sistema nuevo y las deficiencias que tiene el actual para que estos fuera mejorado.

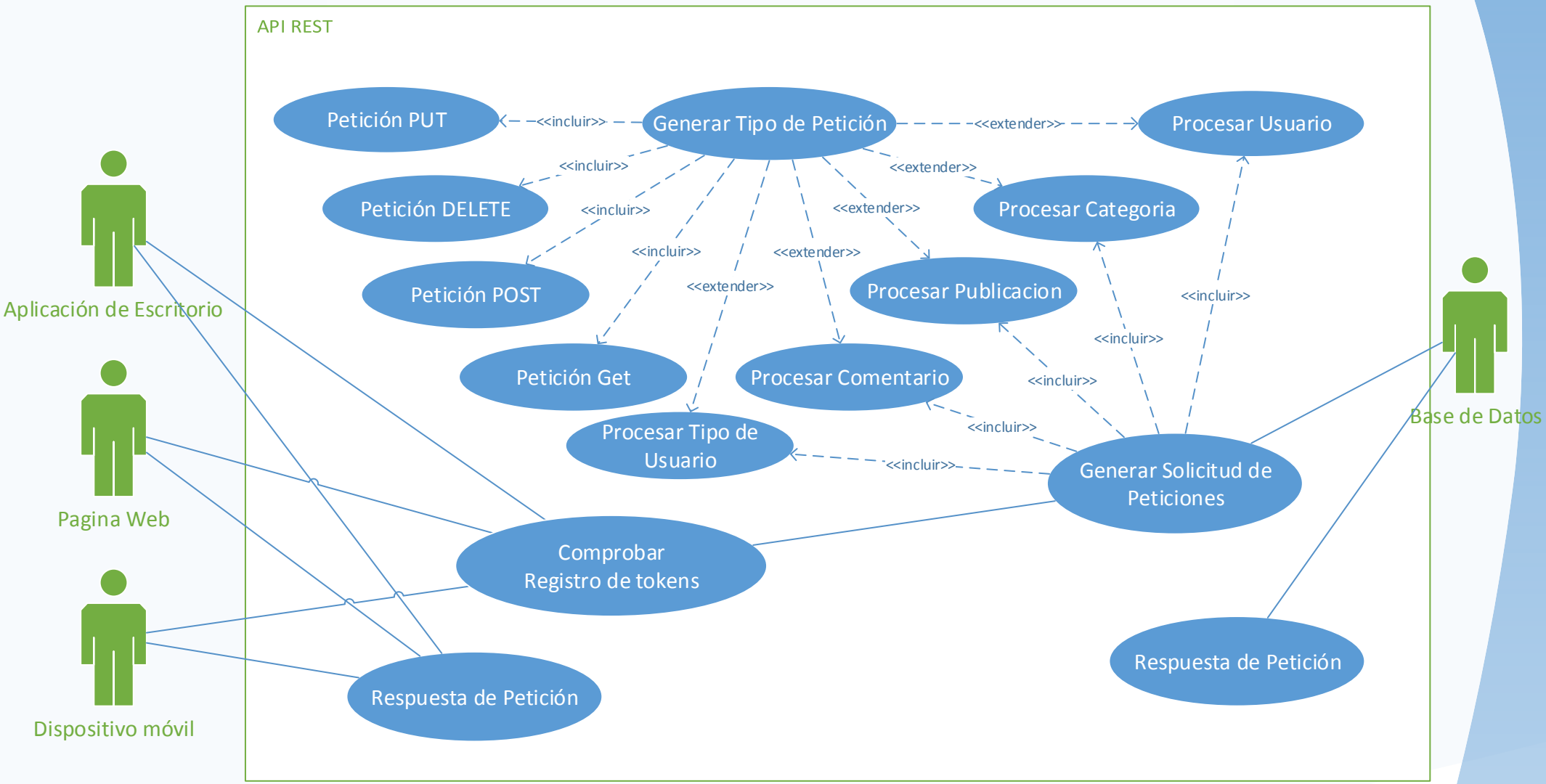
ECONÓMICA (SUFICIENTE DINERO)

- **Costo del tiempo en ser implementado:** (Este proyecto no tiene costo dado que es un proyecto ejecutado por un alumno de la universidad Mariano Gálvez de Guatemala).

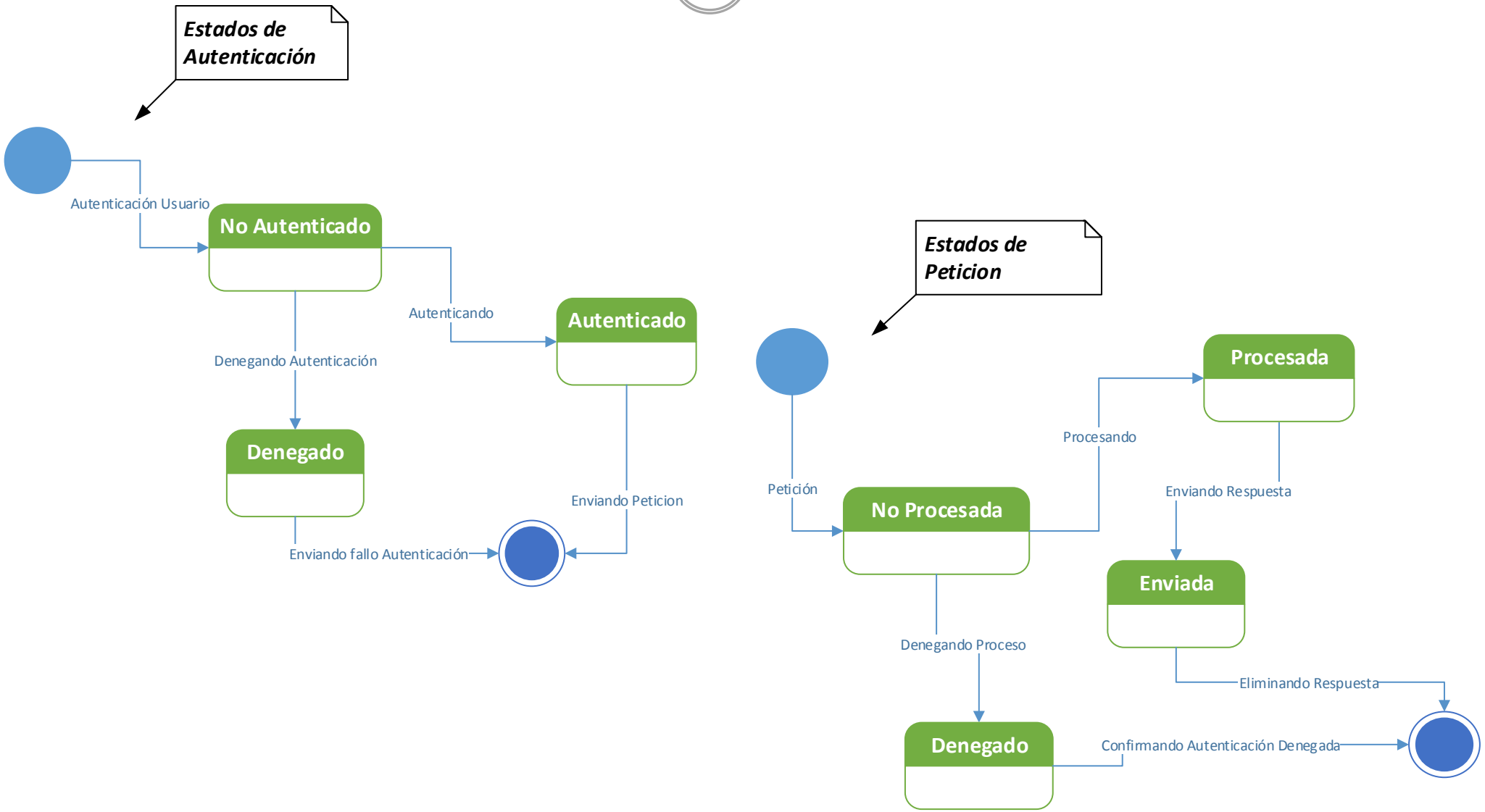
- **Costo del hardware:** este costo no sería necesario teniendo en cuenta el ámbito del proyecto que es universitario se haría con una implementación local.

- **Costo del software:** este programa es gratuito.

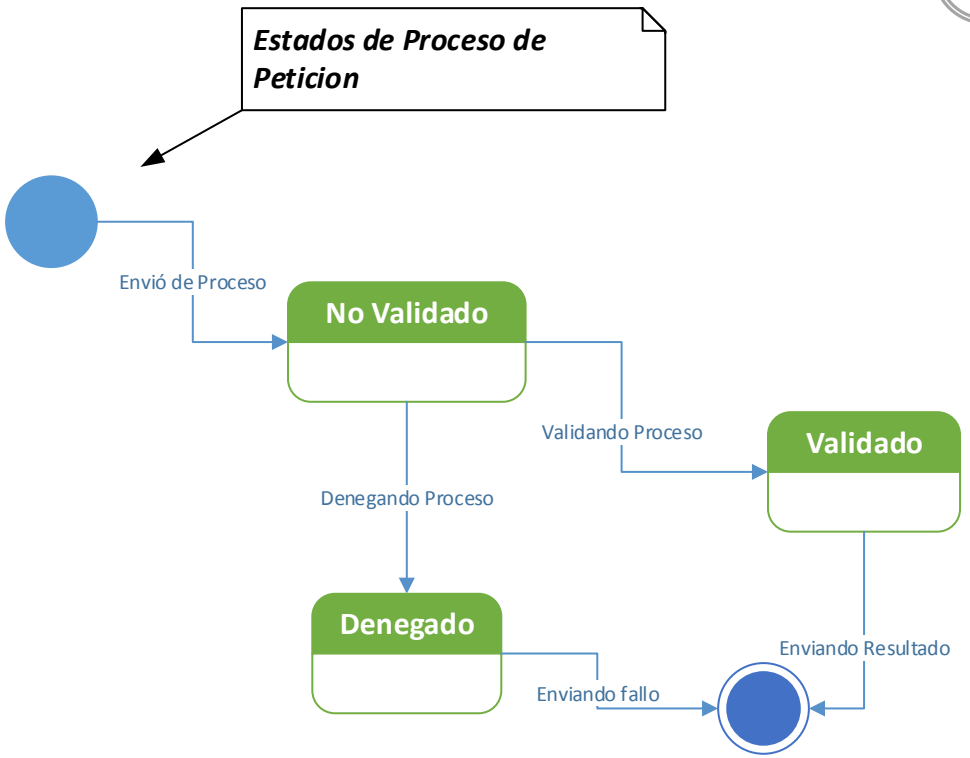
Diagrama de Caso de Uso

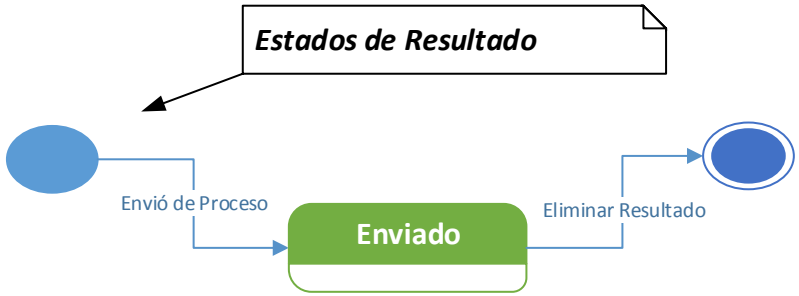


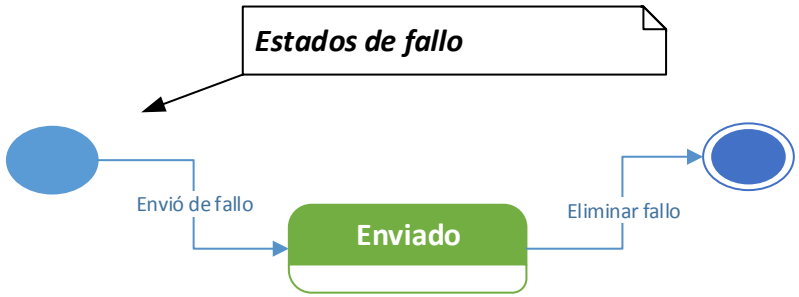
Diagramas de Estados

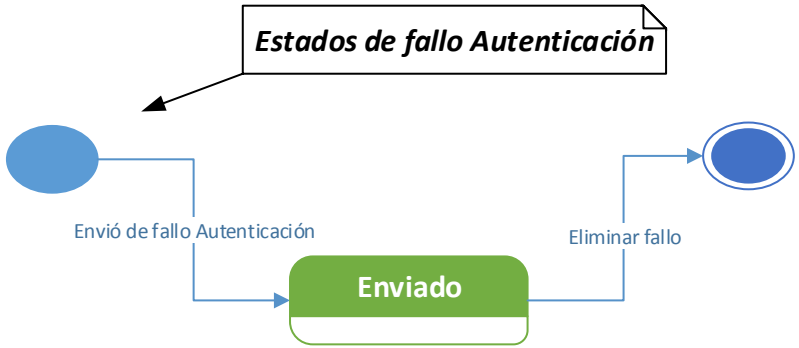


Diagramas de Estados

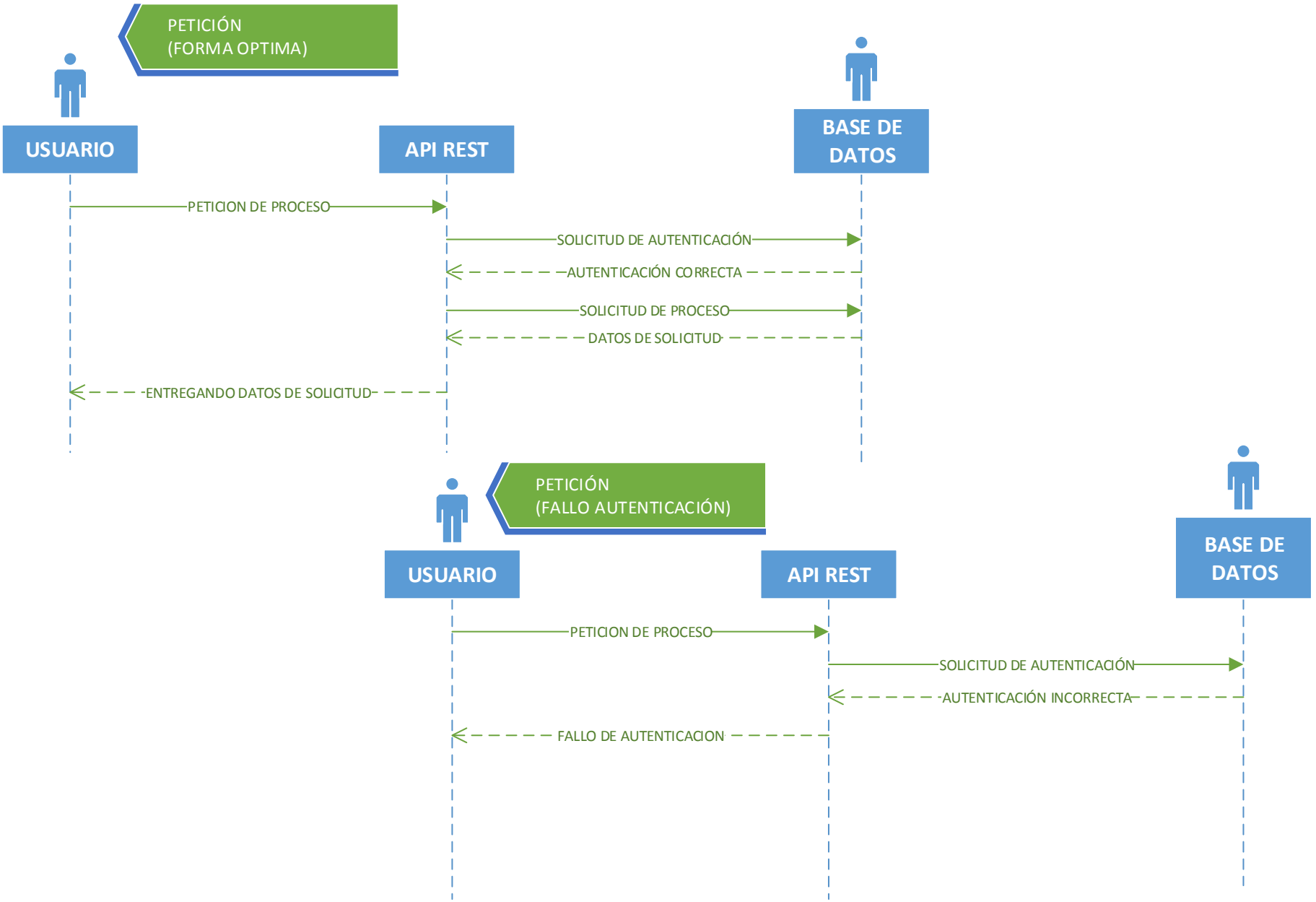








24 de octubre de 2015



24 de octubre de 2015

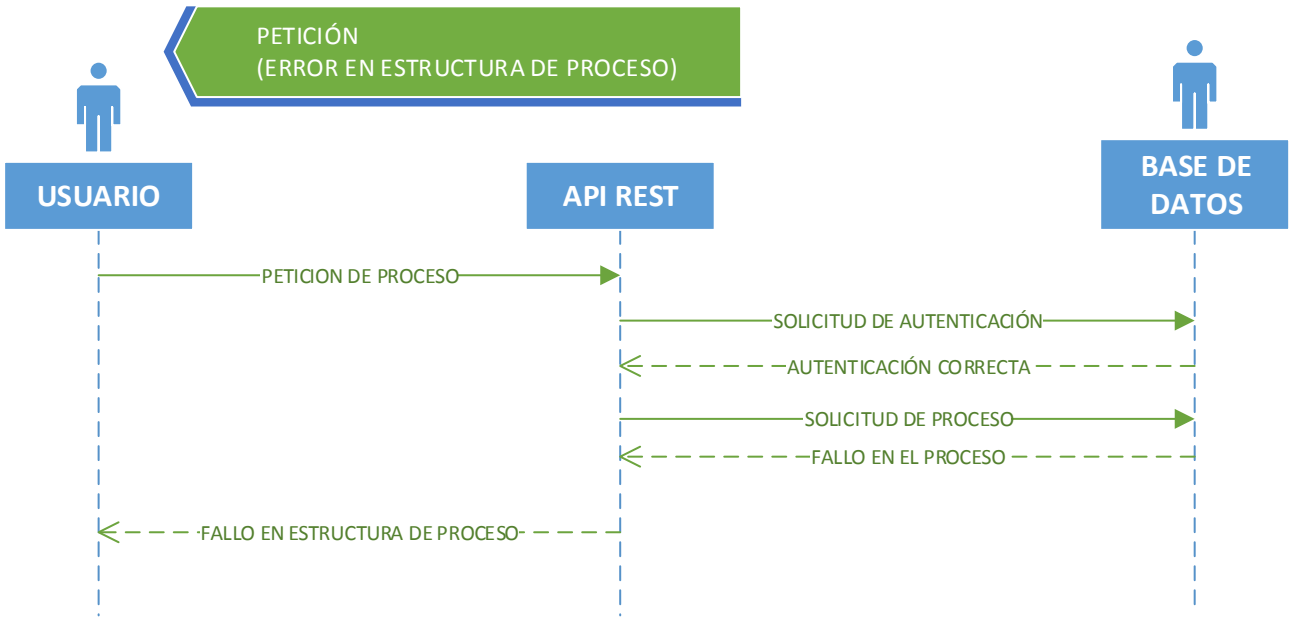
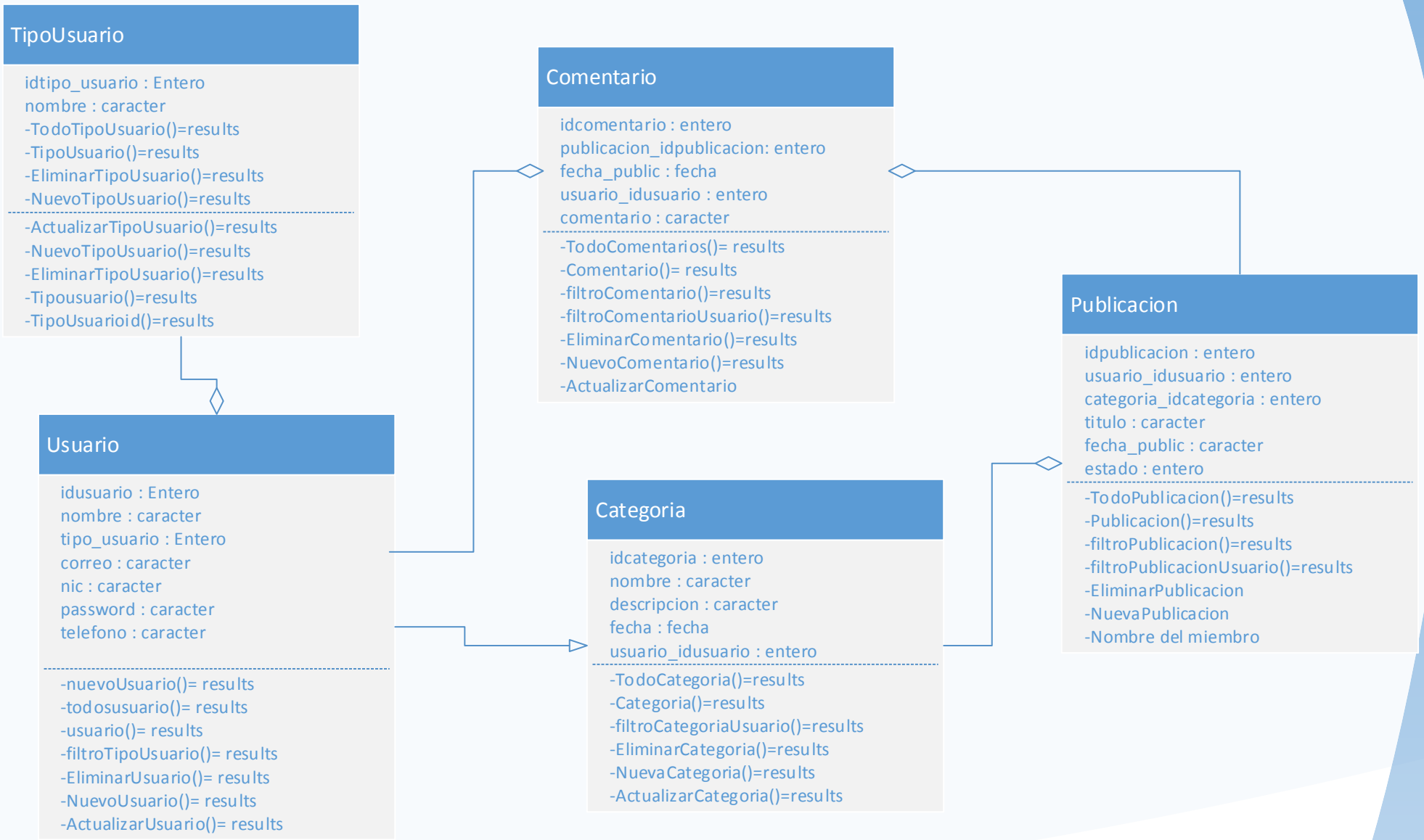


Diagrama de Clases



MODELO ENTIDAD RELACIÓN

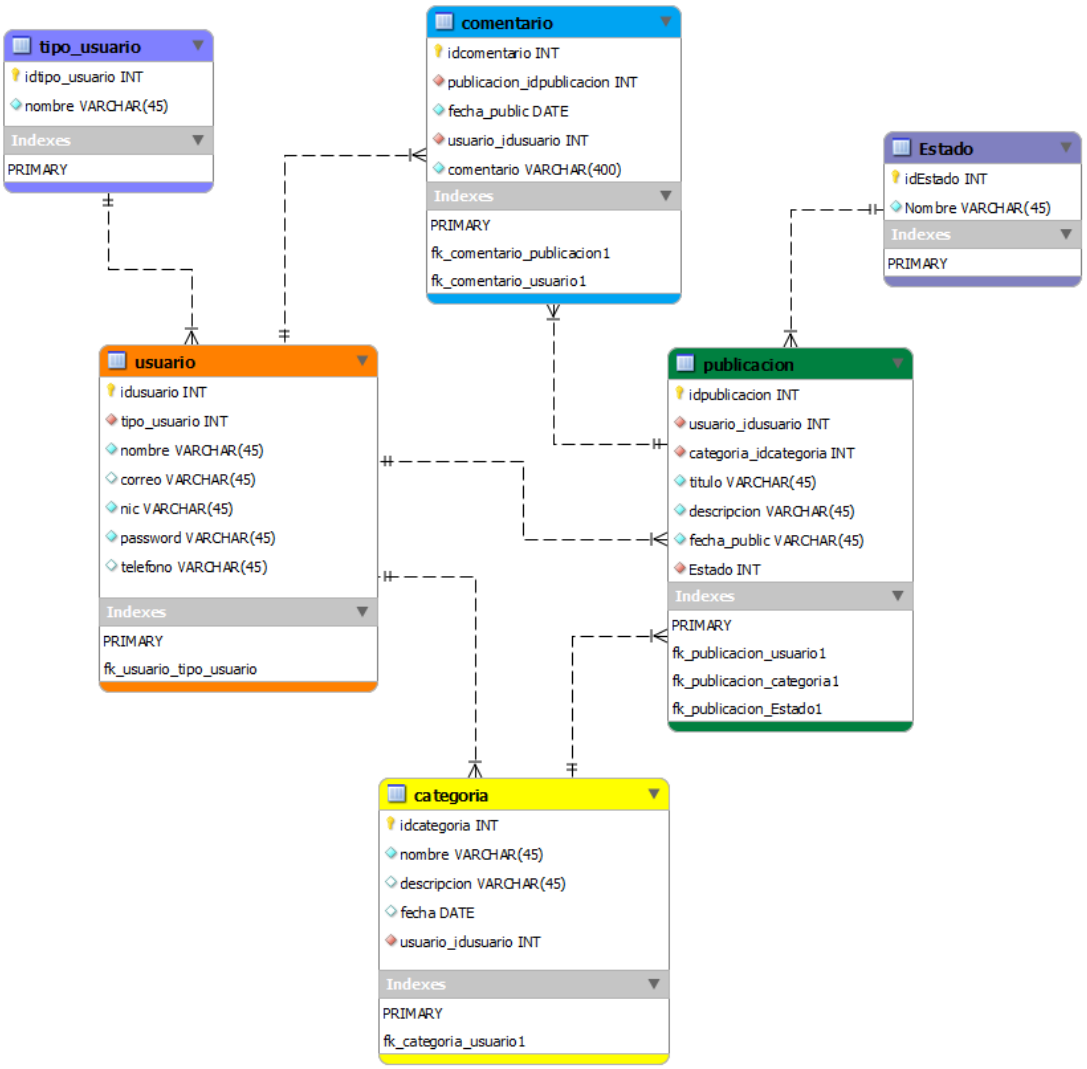
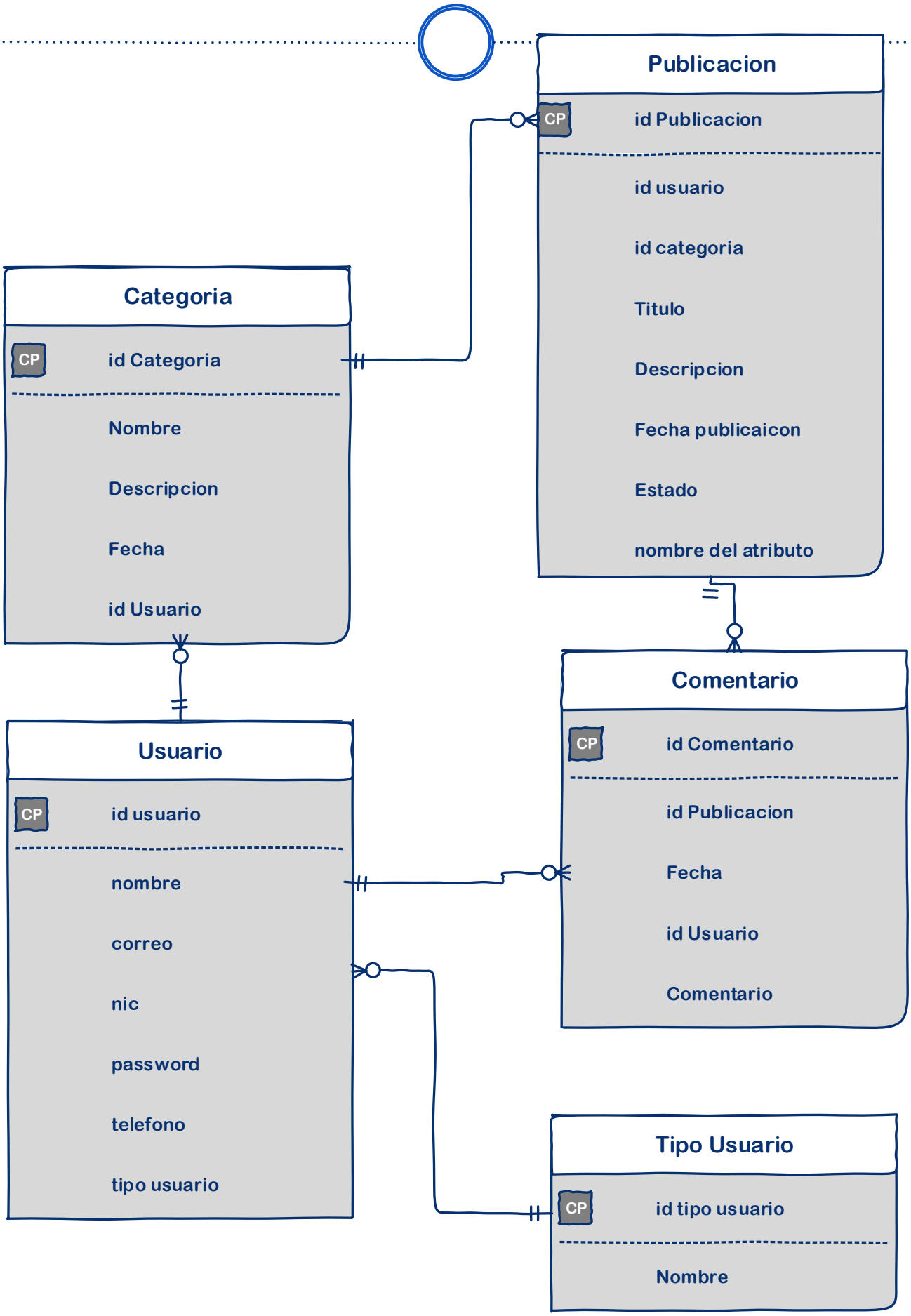


Diagrama Relacional

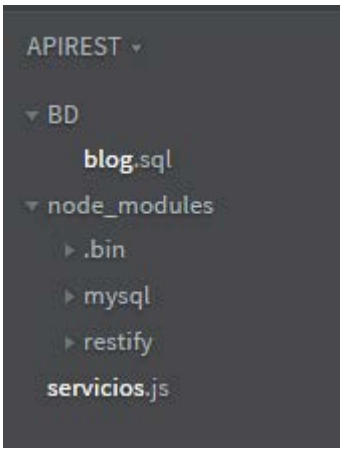


DOCUMENTACIÓN API

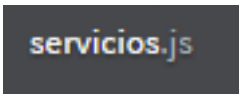
Para comenzar con la documentación de este proyecto se debe tomar en cuenta el manual de instalación de los programas que permitirán el buen funcionamiento de nuestra API REST como lo son (NODE JS) Y (WAMP SERVER), el servidor se puede cambiar por el que gusten, pero debe de tener incorporado o soportar MySQL y PHP que son los lenguajes en que se basó este proyecto.

Lo siguiente es que este proyecto está desarrollado con la ayuda de dos módulos ya desarrollados para NODE JS que son MYSQL y FESTIFY, ya con esto podemos empezar a hablar del código que se utilizó para el desarrollo de esta API REST.

Iniciamos con la estructura de las carpetas todos los documentos que conforman el proyecto están almacenados en la carpeta llamada ***APIREST*** la cual contiene dos carpetas una llamado ***node_modules*** en la cual esta almacenados los módulos que implementamos en este proyecto como lo vimos anterior mente ***MYSQL y FESTIFY.*** y la otra llamada ***BD*** en la cual se encuentra el código de la base de datos de respaldo. Y al final el archivo llamado ***servicios.js*** el cual alojara el código del proyecto APIREST el cual estaría de esta forma.



Ya teniendo la estructura de las carpetas empezaremos a explicar el código de



En estas primeras dos líneas de código importamos los módulos de (RESTIFY) y (MYSQL).

```
1  var restify = require('restify');
2  var mysql = require('mysql');
3
```

En las siguientes líneas de código se genera una variable con la función de MYSQL *crateConnection().*

```
connection = mysql.createConnection({
               host : 'localhost',
               user : 'blog1',
               password : '',
               database: 'blog'
         });
```

Como vemos aquí se le ingresan los datos requeridos para crear una conexión con la base de datos.

En las siguientes líneas de código se crearan variables con las ip y el puerto en donde escuchara la APIREST.

```
var ip_addr = '127.0.0.1';
var port    =  '8080';

var server = restify.createServer
    ({
        name : "API RESTful"
    });
```

Como vemos en el código anterior creamos una variable a la cual le ingresamos el nombre de nuestro servicio API RESTful con la función de RESTIFY *createServer().*

```
server.use(restify.queryParser());
server.use(restify.bodyParser());
server.use(restify.CORS());
```

Se inician los servicios de RESTIFY que nos servirán para el manejo de variables entre peticiones.

```
server.listen(port ,ip_addr,
                            function()
                                {
                                    console.log('%s Escuchando en %s ', server.name , server.url);
                                }
            );
```

Con este código mostraremos en la consola de NODE JS que hemos levantado los servicios de APIREST y que estamos escuchando las peticiones en la IP y en el puerto que especificamos anterior mente.

En este código creamos lo URN (Nombre de Recurso Unitario), para cada petición posible en este caso tenemos definidas las acciones para los Usuarios, Publicaciones, Comentarios, Categorías, Tipos de Usuarios.

```
/*          USUARIOS            */

var PATH = '/usuarios';
var PATH1 = '/nuevo_usuario';
var PATH2 = '/eliminar_usuario';
var PATH3 = '/actulizar_usuario';
```

```
/*      CATEGORIAS          */

var PATH12 = '/categoria';
var PATH13 = '/categoria_nuevo';
var PATH14 = '/categoria_eliminar';
var PATH15 = '/categoria_actualizar';
```

```
/*      TIPO DE USUARIOS          */

var PATH16 = '/tipo_usuario';
var PATH17 = '/tipo_usuario_nuevo';
var PATH18 = '/tipo_usuario_eliminar';
var PATH19 = '/tipo_usuario_actualizar';
```

En el siguiente código veremos las llamadas a las funciones dependiendo del método en el que son llamadas podremos un trozo de código de cada uno GET, POST, DELETE y UPDATE

```
server.get({path : PATH , version : '0.0.1'} , TodosUsuarios);
server.get({path : PATH +'/:idusuario' , version : '0.0.1'} , Usuario);
```

En estas líneas de código vemos dos llamadas del método GET, el cual tiene el PATH que indica la localización, luego la versión de la API REST y por último la selecciona todos los usuarios existentes con la función *TodosUsuarios(),* la siguiente línea muestra lo mismo solo que hace un filtro por el identificador del usuario que ingresamos después de PATH `+'/:idusuario'`.

La función de esta llamada seria esta para *TodosUsuarios().*

```
function TodosUsuarios(req, res, next)
    {
      connection.query('SELECT * FROM `usuario`',
                        function (error, results)
                             {
                                    if(error) throw error;
                                    console.log(results);
                                    res.send(200, results);
                                    return next();
                             }
                       );
    }
```

Esta función requiere los parámetros (req, res) luego llama a la función *connection.query()* la cual posee la sentencia SQL de seleccionar todos los usuarios y una función que registra si sucede algún error en la transacción.

```
function Usuario(req, res, next)
    {
        connection.query('SELECT * FROM `usuario` WHERE `idusuario`='
                         +req.params.idusuario,
                         function (error, results)
                            {
                               if(error) throw error;
                               console.log(results);
                               res.send(200, results);
                               return next();
                            }
                        );
    }
```

Esta es la función usuario la cual selecciona un usuario especifico de la base de datos este código es parecido al anterior lo único que cambia es el WHERE en el cual le indicamos el identificador del usuario que deseamos seleccionar.

En el código siguiente veremos la llamada al método DELETE que se utiliza para eliminar usuarios

```
server.del({path : PATH2 +'/:idusuario', version: '0.0.1'} ,
EliminarUsuario);
```

En estas líneas vemos que el server cambio por del que significa llamar al servicio DELETE después asigna el PATH correspondiente a la operación que en este caso sería el de usuario se le agrega el identificador del usuario que deseamos eliminar más la versión de la API REST y por ultimo llamamos a la función *EliminarUsuario()* que veremos en el código siguiente

```
function EliminarUsuario(req , res , next)
    {

        connection.query('DELETE FROM `usuario` WHERE `idusuario` = '
                         +req.params.idusuario,
                         function (error, success)
                            {
                                if(error) throw error;
                                res.send(200, 'Eliminado con exito');
                            }
                        );
    }
```

Esta función requiere el identificador del usuario a eliminar y lo ingresa en la sentencia SQL que tiene la función *connection.query* () con la variable `+req.params.idusuario` y el mismo procedimiento para eliminar cualquier registro en la base de datos.

En el código siguiente veremos la llamada al método POST el cual nos permitirá insertar usuarios, Publicaciones, Comentarios Categorías y tipos de Usuarios el ejemplo que veremos será insertar usuario.

```
server.post({path : PATH1, version: '0.0.1'} , NuevoUsuario);
```

Como hemos visto cambio la llamada al método POST y el PATH será el correspondiente a Usuario incluimos la versión y llamamos a la función *NuevoUsuario().*

```
function NuevoUsuario(req , res , next)
    {
        var user = {};
        user.tipo_usuario = req.params.tipo_usuario;
        user.nombre = req.params.nombre;
        user.correo = req.params.correo;
        user.nic = req.params.nic;
        user.password = req.params.password;
        user.telefono = req.params.telefono;

        connection.query('INSERT INTO `usuario`(`tipo_usuario`, `nombre`, `correo`, `nic`,
`password`, `telefono`) VALUES(\''
                            +user.tipo_usuario+'\', \''
                            +user.nombre+'\', \''
                            +user.correo+'\', \''
                            +user.nic+'\', \''
                            +user.password+'\', \''
                            +user.telefono+'\', \')'
                            , function (error, success)
                                {
                                    if(error) throw error;
                                     console.log(success);
                                    res.send(200, success.insertId);
                                }
                        );
    }
```

Aquí tenemos el código de la función *NuevoUsuario()*el cual recibe en sus parámetros *req* el cual contiene un array de información enviada en el JSON de la petición, el cual vemos entrando a la función se inicializa un array llamado user y luego se le agregan los datos recolectados por *req que son:*

```
var user = {};
user.tipo_usuario = req.params.tipo_usuario;
user.nombre = req.params.nombre;
user.correo = req.params.correo;
user.nic = req.params.nic;
user.password = req.params.password;
user.telefono = req.params.telefono;
```

Luego de esto llamamos a la función connection.query() la cual tendrá la sentencia SQL de inserción de usuario con cada uno de sus campo a los cuales en el VALUES colocamos las variable correspondientes que creamos anterior mente

```
        connection.query('INSERT INTO `usuario`(`tipo_usuario`, `nombre`, `correo`, `nic`,
`password`, `telefono`) VALUES(\''
```

```
 +user.tipo_usuario+'\', \''
 +user.nombre+'\', \''
 +user.correo+'\', \''
 +user.nic+'\', \''
 +user.password+'\', \''
 +user.telefono+'\', \')'
```

Y como siempre en toda función llamamos al callback que nos indicara si se encontró un error en el procedimiento.

```
function (error, success)
  {
      if(error) throw error;
       console.log(success);
      res.send(200, success.insertId);
  }
```

Si todo resulta bien se nos indicara con res.send() una confinación http OK (código 200) y el número de identificación del nuevo usuario.

En el próximo código veremos la llamada al método PUT el cual no permitirá actualizar un usuario, comentario, Publicación, Categoría y tipo de Usuario esta es la llamada.

```
server.put({path : PATH3, version: '0.0.1'} , ActualizarUsuario);
```

Como hemos visto anterior mente lo que cambia es el método en este caso put luego el PATH correspondiente a que debemos actualizar luego el identificador a modificar, la versión y por último la función que se ejecutara.

```
function ActualizarUsuario(req , res , next)
    {
        var user = {};
        user.tipo_usuario = req.params.tipo_usuario;
        user.nombre = req.params.nombre;
        user.correo = req.params.correo;
        user.nic = req.params.nic;
        user.password = req.params.password;
        user.telefono = req.params.telefono;
        connection.query('UPDATE `usuario` SET (`tipo_usuario`= \''
                            +user.tipo_usuario+'\', `nombre`= \''
                            +user.nombre+'\', `correo`= \''
                            +user.correo+'\', `nic`= \''
                            +user.nic+'\', `password`= \''
                            +user.password+'\',`telefono`= \''
                            +user.telefono+'\', \') WHERE `idusuario` = '+req.params.idusuario
                            , function (error, success)
                                    {
                                        if(error) throw error;
                                         console.log(success);
                                        res.send(200, success);
                                    }
                        );
    }
```

Esta es la función ActualizarUsuario() el cual recibe de parámetro todos los datos que se van a modificar según el identificador ingresado en WHERE de la sentencia SQL si todo está bien el en ejecución de la sentencia nos debería actualizar en este caso en la tabla usuario al usuario identificado.

CONCLUSIONES

RECOMENDACIONES

BIBLIOGRAFÍA

- http://www.nodebeginner.org/index-es.html
- http://restify.com/#routing
- http://restify.com/#creating-a-server
- http://restify.com/#httpclient
- http://restify.com/#stringclient
- http://restify.com/#client-api
- http://restify.com/#jsonclient
- https://hackpad.com/ep/pad/static/UnS1TzLP5jh
- https://es.wikipedia.org/wiki/Representational_State_Transfer
- https://es.wikipedia.org/wiki/Simple_Object_Access_Protocol
- http://www.ics.uci.edu/~fielding/pubs/dissertation/
- https://msdn.microsoft.com/es-gt/library/dd409437.aspx
- http://elvex.ugr.es/decsai/java/pdf/3C-Relaciones.pdf
- http://users.dcc.uchile.cl/~psalinas/uml/modelo.html
- https://www.google.com.gt/webhp?sourceid=chrome-instant&ion=1&espv=2&ie=UTF-8#q=documentacion+de+restify
- https://www.google.com.gt/webhp?sourceid=chrome-instant&ion=1&espv=2&ie=UTF-8#q=documentacion+de+mysql+js
- https://www.desarrolloweb.com/que_es_REST.html
- http://www.genbetadev.com/frameworks/introduccion-a-la-programacion-asincrona-con-nodejs-ii
- http://www.genbetadev.com/frameworks/como-funciona-node-js
- http://stackoverflow.com/questions/22208975/restify-on-node-js-post-body-json
- http://stackoverflow.com/questions/19724281/restify-2-6-post-data
- http://ciclodevidasoftware.wikispaces.com/CICLO+DE+VIDA+EN+ESPIRAL
- https://es.wikipedia.org/wiki/Desarrollo_en_espiral
- http://modeloespiral.blogspot.com/
- https://es.wikipedia.org/wiki/Proceso_Unificado_de_Rational
- https://softwarerecopilation.wordpress.com/modelo-rup/
- http://procesoderational.blogspot.com/2009/12/ciclo-de-vida-de-rup.html
- http://comunidadilgo.org/contenido/portal/portaldoc453_3.pdf?0e5f0034b0d9823ead8694fb2a4ff2bb
- http://ciclodevidasoftware.wikispaces.com/CICLO+DE+VIDA+EN+ESPIRAL
- https://es.wikipedia.org/wiki/Desarrollo_en_espiral
- http://modeloespiral.blogspot.com/
- https://es.wikipedia.org/wiki/Proceso_Unificado_de_Rational
- https://softwarerecopilation.wordpress.com/modelo-rup/
- http://procesoderational.blogspot.com/2009/12/ciclo-de-vida-de-rup.html
- http://comunidadilgo.org/contenido/portal/portaldoc453_3.pdf?0e5f0034b0d9823ead8694fb2a4ff2bb
- http://restify.com/
- http://www.wampserver.com/en/

GLOSARIO

Análisis:

El análisis de sistemas es la ciencia encargada del análisis de sistemas grandes y complejos, y la interacción entre los mismos. Esta área se encuentra muy relacionada con la Investigación operativa. También se denomina análisis de sistemas a una de las etapas de construcción de un sistema informático, que consiste en relevar la información actual y proponer los rasgos generales de la solución futura.

Api (Application programming interface):

En la programación de computadoras, una interfaz de programación de aplicaciones (API) es un conjunto de rutinas, protocolos y herramientas para la construcción de aplicaciones de software.

Aplicación web:

En la ingeniería de software se denomina aplicación web a aquellas herramientas que los usuarios pueden utilizar accediendo a un servidor web a través de Internet o de una intranet mediante un navegador. En otras palabras, es una aplicación software que se codifica en un lenguaje soportado por los navegadores web en la que se confía la ejecución al navegador.

Aplicación de Escritorio o Entorno de Escritorio:

Entorno de escritorio (en inglés desktop environment, abreviado DE) es un conjunto de software para ofrecer al usuario de una computadora una interacción amigable y cómoda. Es una implementación de interfaz gráfica de usuario que ofrece facilidades de acceso y configuración, como barras de herramientas e integración entre aplicaciones con habilidades como arrastrar y soltar. Los entornos de escritorios por lo general no permiten el acceso a todas las características que se encuentran en un sistema operativo, por la ausencia de una interfaz gráfica. En su lugar, la tradicional interfaz de línea de comandos (CLI) todavía se utiliza cuando el control total sobre el sistema operativo se requiere en estos casos.

Calidad:

Conjunto de propiedades inherentes a una cosa que permite caracterizarla y valorarla con respecto a las restantes de su especie.

Callback:

En programación de computadoras, una devolución de llamada o retro llamada (en inglés: callback) es una función "A" que se usa como argumento de otra función "B". Cuando se llama a "B", ésta ejecuta "A". Para conseguirlo, usualmente lo que se pasa a "B" es el puntero a "A".

Cliente:

Persona que utiliza los servicios de un profesional o de una empresa, especialmente la que lo hace regularmente.

CRUD:

En computación CRUD es el acrónimo de Crear, Obtener, Actualizar y Borrar (del original en inglés: Create, Read, Update and Delete). Se usa para referirse a las funciones básicas en bases de datos o la capa de persistencia en un software.

Desarrollo:

Programación de los sistemas de cómputo

Diseño:

Como un proceso o labor a , proyectar, coordinar, seleccionar y organizar un conjunto de elementos para producir y crear objetos visuales destinados a comunicar mensajes específicos a grupos determinados.

Dispositivos:

Pieza o conjunto de piezas o elementos preparados para realizar una función determinada y que generalmente forman parte de un conjunto más complejo.

Dispositivo móvil:

Los dispositivos móviles son aparatos de pequeño tamaño, con algunas capacidades de procesamiento, con conexión permanente o intermitente a una red, con memoria limitada, diseñados específicamente para una función, pero que pueden llevar a cabo otras funciones más generales.

Escalable:

En telecomunicaciones y en ingeniería informática, la escalabilidad es la propiedad deseable de un sistema, una red o un proceso, que indica su habilidad para reaccionar y adaptarse sin perder calidad, o bien manejar el crecimiento

continuo de trabajo de manera fluida, o bien para estar preparado para hacerse más grande sin perder calidad en los servicios ofrecidos.

Flexible:

Respecto a la calidad de un programa, la flexibilidad hace referencia a establecer en qué medida el programa es susceptible de ser cambiado.

Glosario

Catálogo alfabetizado de las palabras y expresiones de uno o varios textos que son difíciles de comprender, junto con su significado o algún comentario.

HARDWARE:

Conjunto de elementos físicos o materiales que constituyen una computadora o un sistema informático.

HOST:

El término host ("anfitrión", en español) es usado en informática para referirse a las computadoras conectadas a una red, que proveen y utilizan servicios de ella. Los usuarios deben utilizar anfitriones para tener acceso a la red.

HTTP:

Es un protocolo orientado a transacciones y sigue el esquema petición-respuesta entre un cliente y un servidor. Al cliente que efectúa la petición (un navegador web o un spider) se lo conoce como "user agent" (agente del usuario). A la información transmitida se la llama recurso y se la identifica mediante un localizador uniforme de recursos (URL). El resultado de la ejecución de un programa, una consulta a una base de datos, la traducción automática de un documento, etc.

Ingeniería:

La ingeniería es el conjunto de conocimientos y técnicas científicas, empíricas y prácticas aplicadas a la invención, el diseño, el desarrollo, la construcción, el mantenimiento y el perfeccionamiento de tecnologías, estructuras, máquinas, herramientas, sistemas, materiales y procesos para la resolución de problemas prácticos.

Ingeniería en sistemas:

La ingeniería de sistemas es un modo de enfoque interdisciplinario que permite estudiar y comprender la realidad, con el propósito de implementar u optimizar sistemas complejos. Puede también verse como la aplicación

tecnológica de la teoría de sistemas a los esfuerzos de la ingeniería, adoptando en todo este trabajo el paradigma sistémico. La ingeniería de sistemas integra otras disciplinas y grupos de especialidad en un esfuerzo de equipo, formando un proceso de desarrollo centrado.

Ingeniería de software:

Ingeniería de software es la aplicación de un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación y mantenimiento de software,1 y el estudio de estos enfoques, es decir, la aplicación de la ingeniería al software.2 Integra matemáticas, ciencias de la computación y prácticas cuyos orígenes se encuentran en la ingeniería

JSON:

Acrónimo de JavaScript Object Notation, es un formato ligero para el intercambio de datos. JSON es un subconjunto de la notación literal de objetos de JavaScript que no requiere el uso de XML

MySQL:

Es un sistema de gestión de bases de datos relacional, multihilo y multiusuario con más de seis millones de instalaciones. MySQL AB —desde enero de 2008 una subsidiaria de Sun Microsystems y ésta a su vez de Oracle Corporation desde abril de 2009— desarrolla MySQL como software libre en un esquema de licenciamiento dual.

NODE JS:

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

OSI:

El modelo de interconexión de sistemas abiertos (ISO/IEC 7498-1), más conocido como “modelo OSI”, (en inglés, Open System Interconnection) es un modelo de referencia para los protocolos de red la arquitectura en capas, creado en el año 1980 por la Organización Internacional de Normalización (ISO, International Organization for Standardization).

PHP:

PHP es un lenguaje de programación de uso 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. El código es interpretado por un servidor web con un módulo de procesador de PHP que genera la página Web resultante. PHP ha evolucionado por lo que ahora incluye también una interfaz de línea de comandos que puede ser usada en aplicaciones gráficas independientes. Puede ser usado en la mayoría de los servidores web al igual que en casi todos los sistemas operativos y plataformas sin ningún costo.

Plataforma:

En informática, una plataforma es un sistema que sirve como base para hacer funcionar determinados módulos de hardware o de software con los que es compatible. Dicho sistema está definido por un estándar alrededor del cual se determina una arquitectura de hardware y una plataforma de software (incluyendo entornos de aplicaciones). Al definir plataformas se establecen los tipos de arquitectura, sistema operativo, lenguaje de programación o interfaz de usuario compatibles.

REST (Representational state transfer):

En la computación, Representational State Transfer (REST) es el estilo arquitectónico de software de la World Wide Web. REST da un conjunto coordinado de las limitaciones en el diseño de componentes en un distribuido hipermedia sistema que puede conducir a un mayor rendimiento y más fácil de mantener la arquitectura.

RESTIFI:

Es un módulo Node.js construido específicamente para que pueda construir servicios web REST correctas. Es intencionadamente inspira en gran medida de expreso, ya que es más o menos la API de facto para escribir aplicaciones web en la parte superior de node.js.

RUP:

El Proceso Racional Unificado (Rational Unified Process en inglés, habitualmente resumido como RUP) es un proceso de desarrollo de software desarrollado por la empresa Rational Software, actualmente propiedad de IBM. Junto con el Lenguaje Unificado de Modelado UML, constituye la metodología estándar más utilizada para el análisis, diseño, implementación y documentación de sistemas orientados a objetos.

Servicio:

Un servicio es un conjunto de actividades que buscan responder a las necesidades de un cliente.

Servicio web:

Es una tecnología que utiliza 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.

Servidor web:

Un servidor web o servidor HTTP es un programa informático que procesa una aplicación del lado del servidor, realizando conexiones bidireccionales y/o unidireccionales y síncronas o asíncronas con el cliente y generando o cediendo una respuesta en cualquier lenguaje o Aplicación del lado del cliente. El código recibido por el cliente suele ser compilado y ejecutado por un navegador web. Para la transmisión de todos estos datos suele utilizarse algún protocolo. Generalmente se usa el protocolo HTTP para estas comunicaciones, perteneciente a la capa de aplicación del modelo OSI. El término también se emplea para referirse al ordenador que ejecuta el programa

SOFTWARE:

Conjunto de programas y rutinas que permiten a la computadora realizar determinadas tareas.

SOUP:

SOAP (siglas de Simple Object Access Protocol) es un protocolo estándar que define cómo dos objetos en diferentes procesos pueden comunicarse por medio de intercambio de datos XML. Este protocolo deriva de un protocolo creado por Dave Winer en 1998, llamado XML-RPC. SOAP fue creado por Microsoft, IBM y otros. Está actualmente bajo el auspicio de la W3C. Es uno de los protocolos utilizados en los servicios Web.

Tokens:

Pequeños grupos de datos que representan un conjunto de información mayor previamente establecida.

URI:

Un identificador de recursos uniforme o URI —del inglés Uniform Resource Identifier— es una cadena de caracteres que identifica los recursos de una red de forma unívoca. La diferencia respecto a un localizador de recursos uniforme (URL) es que estos últimos hacen referencia a recursos que, de forma general, pueden variar en el tiempo.

URL:

Un localizador de recursos uniforme (conocido por la sigla URL, del inglés Uniform Resource Locator) es un identificador de recursos uniforme (Uniform Resource Identifier, URI) cuyos recursos referidos pueden cambiar, esto es, la dirección puede apuntar a recursos variables en el tiempo.1 Están formados por una secuencia de caracteres, de acuerdo a un formato modélico y estándar, que designa recursos en una red, como internet.

URN:

URN es un acrónimo inglés de Uniform Resource Name, en español "Nombre de recurso uniforme". Un URN funciona de manera similar a un URL. Éstos identifican recursos en la web, pero a diferencia de un URL, no indican exactamente dónde se encuentra ese objeto. Básicamente un URI = URL + URN

Wamp server:

Es un entorno de desarrollo web de Windows. Te permite crear aplicaciones web con Apache 2, PHP y una base de datos MySQL. Al lado, PhpMyAdmin permite administrar fácilmente sus bases de datos.

World Wide Web

En informática, la World Wide Web (WWW) o red informática mundial es un sistema de distribución de documentos de hipertexto interconectados y accesibles vía Internet. Con un navegador web, un usuario visualiza sitios web compuestos de páginas web que pueden contener texto, imágenes, vídeos u otros contenidos multimedia, y navega a través de esas páginas usando hiperenlaces.

XML:

XML, siglas en inglés de eXtensible Markup Language ('lenguaje de marcas extensible'), es un lenguaje de marcas desarrollado por el World Wide Web Consortium (W3C) utilizado para almacenar datos en forma legible. Deriva del lenguaje SGML y permite definir la gramática de lenguajes específicos (de la misma manera que HTML es a su vez un lenguaje definido por SGML) para estructurar documentos grandes.