# Proyecto de Análisis, Diseño y Desarrollo

## Página web
## Implementación de API REST

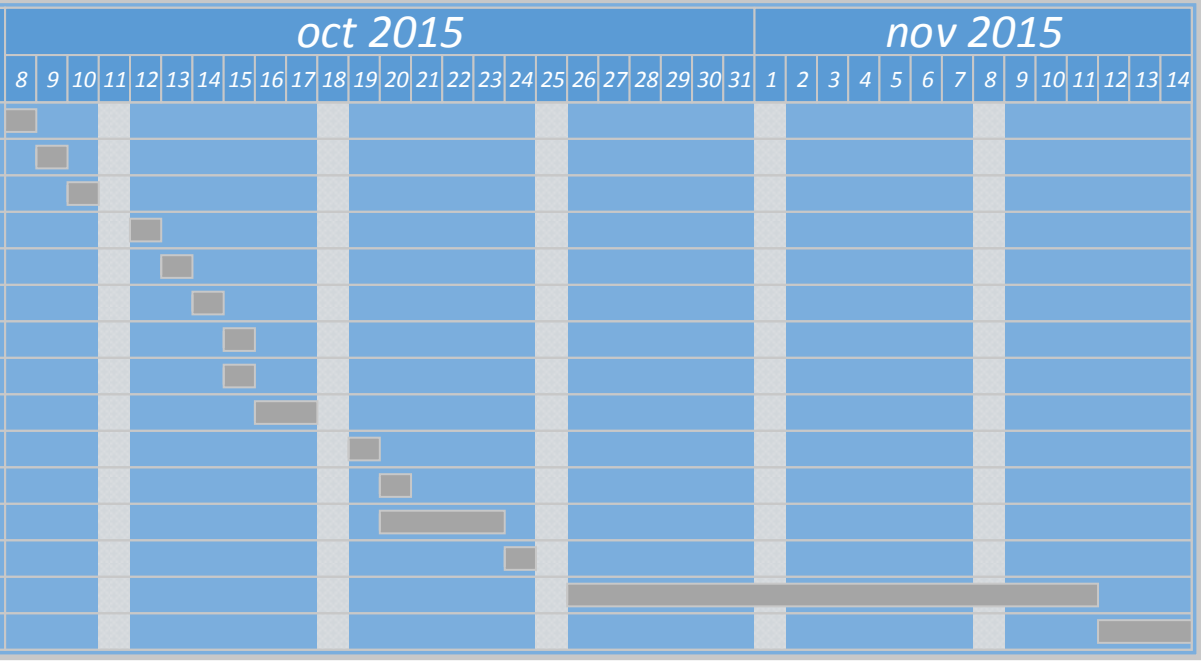
19/11/2015
**Utilizando servicios a través de una API REST**
Rudy Donaldo Marroquín García
1490-09-9429

Ing. Iván Antonio de león Fuentes
Evaluación Privada

# INDICE

# 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 |



# 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 |

# INVESTIGACIÓN PRELIMINAR

ANÁLISIS DEL PROBLEMA Y OPORTUNIDADES

**Nombre de la Empresa:**

El mega blog

**Descripción del Problema:**

Se necesita crear un blog en el cual se pueda construir y distribuir la información publicada por los usuarios en este sitio de forma eficiente y agradable para el usuario que plasma sus ideas o conocimiento adquirido, como para los que leerán el blog, se necesita que este pueda ser accedido por cualquier dispositivo (teléfono, Tablet, computadoras, etc.), se necesita que la información contenida en este pueda ser extraída de forma consistente.

**Posible Solución:**

Atreves de herramienta informáticas se puede realizar una página web total mente responsiva esto quiere decir que es adaptable a cualquier dispositivo móvil o de escritorio pueda acceder a ella, se puede implementara una API la cual ofrecerá seguridad en la información y resguardar la información personal de los usuarios de este sitio.

**Objetivos:**

Generales y Específicos:

- Crear un medio de comunicación cómodo y fácil de utilizar permitiendo a sus usuarios una experiencia agradable.
- Transmitir información de forma segura y confiable.
- Ser una herramienta que se acomoda a las necesidades del usuario.
- Tener una herramienta de información disponible en cualquier momento y en cualquier dispositivo solo con tener acceso a internet.

# ANÁLISIS DE REQUERIMIENTOS

## DETERMINACIÓN DE REQUERIMIENTOS

**¿Qué es lo que quiere el cliente?**

Contar con un sistema de almacenamiento y transmisión de información en el cual puedan manejar dicha información de forma fácil y agradable, los usuarios podrán agregar información nueva, cada publicación de los usuarios podrán ser comentadas por otros usuarios y el mismo, como respuesta a estos comentarios para crear un tipo de conversación o chat para hacer interactiva cada publicación.

## ESPECIFICACIÓN DE REQUERIMIENTOS

**¿Cuál es el proceso básico?**

El proceso básico de la organización es contar con una base de datos de los alumnos y personas encargadas de este, donde los pueda clasificar de diversas formas y asignar un orden a estos.

**¿Qué datos utiliza o produce este proceso?**

*Clientes:*
- Nombres
- Apellidos
- Genero
- Edad
- Lugar de nacimiento
- Grado académico
- Carné (Generado por el sistema)
- Nombre de los padres de familia
- Nombre del encargado
- Fecha de inscripción del alumno
- Escuela o Colegio de donde procede
- Código personal establecido por el MINEDUC

*Servicios*
- Nombres
- Apellidos
- DPI
- Teléfono
- Correo electrónico

# CICLO DE VIDA

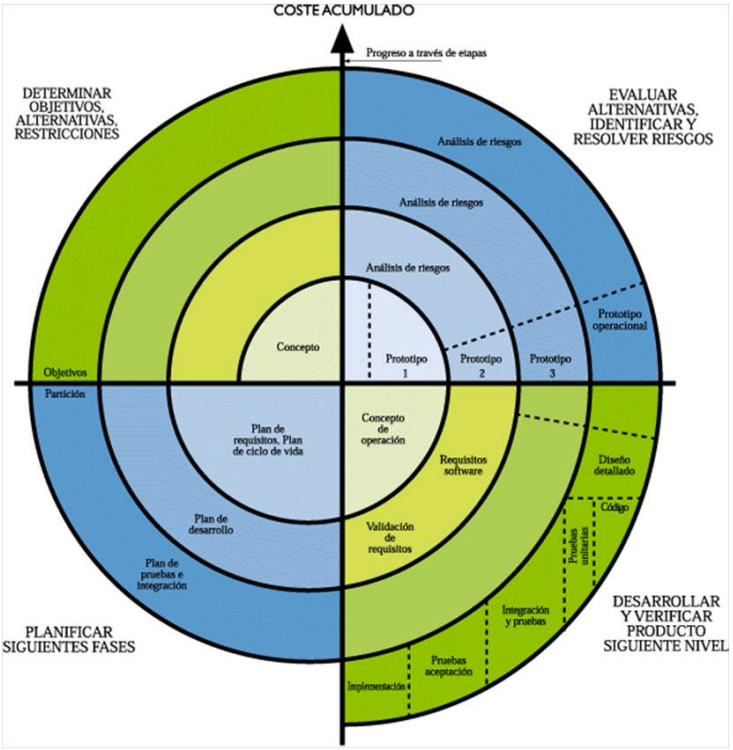
## ESPIRAL

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

**Inicio**

El objetivo general de esta fase es establecer un acuerdo entre todos los interesados acerca de los objetivos del proyecto.

**Elaboración**

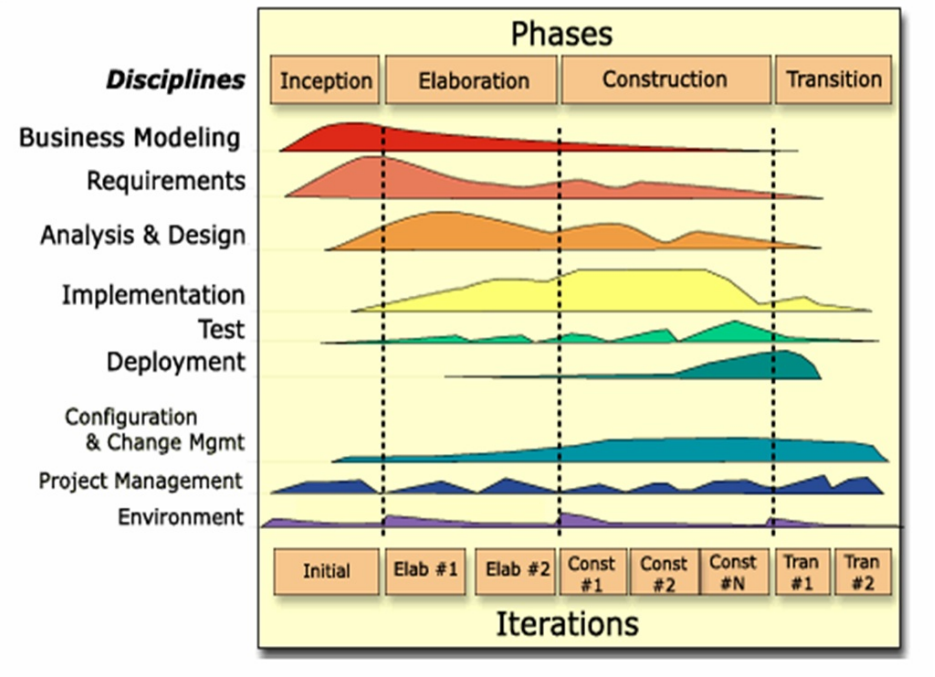
El objetivo en esta fase es establecer la arquitectura base del sistema para proveer bases estables para el esfuerzo de diseño e implementación en la siguiente fase.

**Construcción**

El objetivo de la fase de construcción es clarificar los requerimientos faltantes y completar el desarrollo del sistema basados en la arquitectura base.

**Transición**

Esta fase se enfoca en asegurar que el software esté disponible para sus usuarios.



# 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 hace 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

Enlace información: ics.uci.edu

## ¿Cómo conseguimos que las comunicaciones cliente servidor sean lo más eficaces posibles?

## 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.

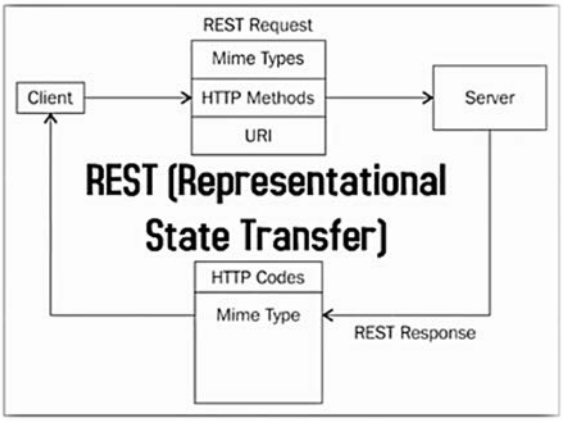
5. Comunicaciones sin estado, no se pueden mantener ningún tipo de persistencia en las transacciones, el servidor trata cada transacción como algo total mente independiente y la comunicación siempre devuelve una estado final.

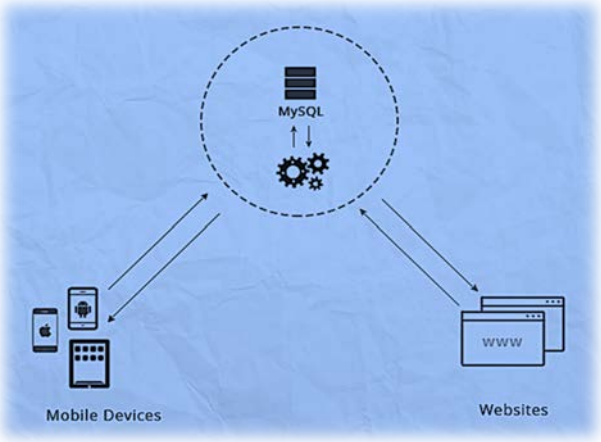
# Arquitectura de API REST

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 o operaciones que realiza cada transacción es diferente y separada aunque las peticiones sean parecidas el 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 token 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 son idempotentes.
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. Puede encontrar la lista de estos tipos MIME aquí.

## API Key

Si está desarrollando una API privada donde desea restringir el acceso o limitarlo según convenga, lo mejor es proteger la API utilizando una API Key, para ello podría apoyarse con OAuth2 u otro según vea por conveniente.

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

Authorization: cfb7acfbb9bb8...

# ANÁLISIS DE FACTIBILIDAD

## TÉCNICA (CONOCIMIENTO)

- 2 Computadoras con características en hardware y software suficientes para implementar un sistema.
- Impresoras.
- Acceso a internet.

## OPERACIONAL (SUFICIENTE PERSONAL)

- **¿Existe apoyo suficiente para el proyecto por parte de la administración?** El proyecto está respaldado por la administración ya que facilitara el acceso a la información de datos personales y financiaros de los alumnos. trabaje, esto hace que el sistema sea

- **Los métodos que actualmente se usan en la empresa, ¿son aceptados por los usuarios?** Son aceptados, pero porque no se había dado la oportunidad de que ellos adquirieran un sistema nuevo para manejar los datos. Pero con la propuesta dada se ven las deficiencias de este sistema.

- **¿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)

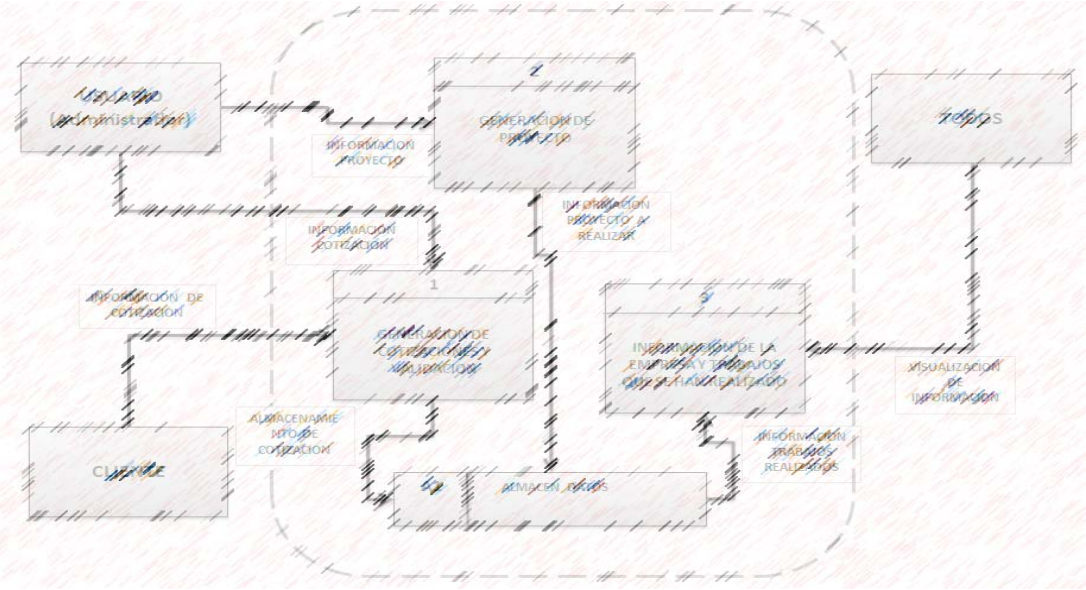
- **Costo del tiempo en ser implementado**(Pendiente)

- **Costo del hardware:** este costo no sería necesario ya que la empresa cuenta con el equipo necesario para implementar el sistema.

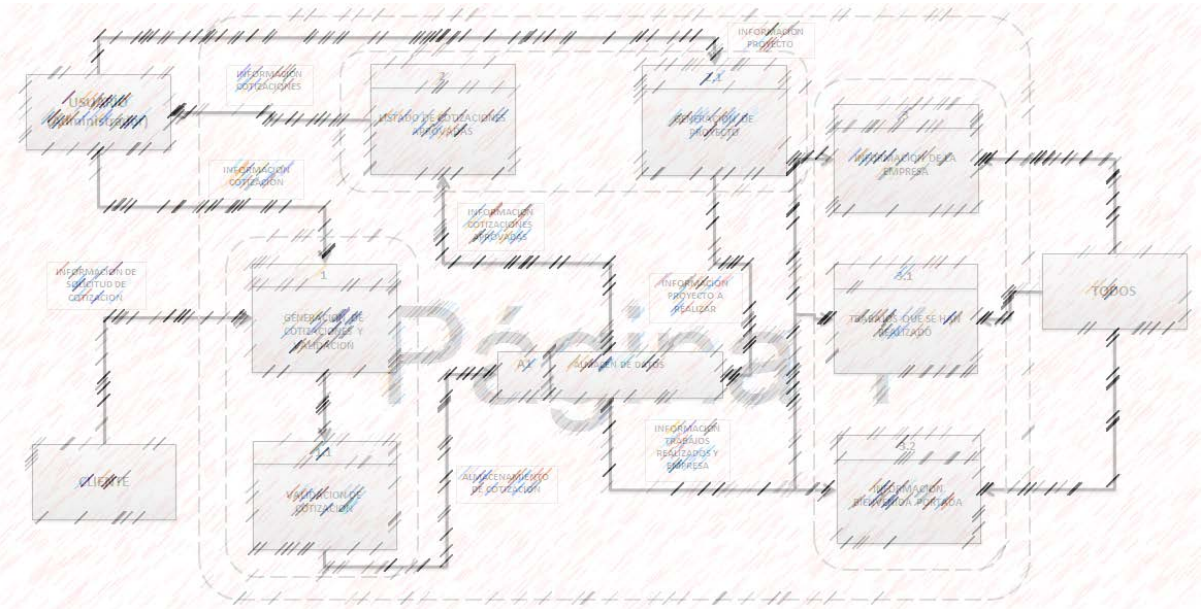
- **Costo del software:** La empresa está dispuesta a invertir en esta área, ya que para ellos es impórtate la seguridad y la accesibilidad de los datos.

# DIAGRAMA DE FLUJO DE DATOS

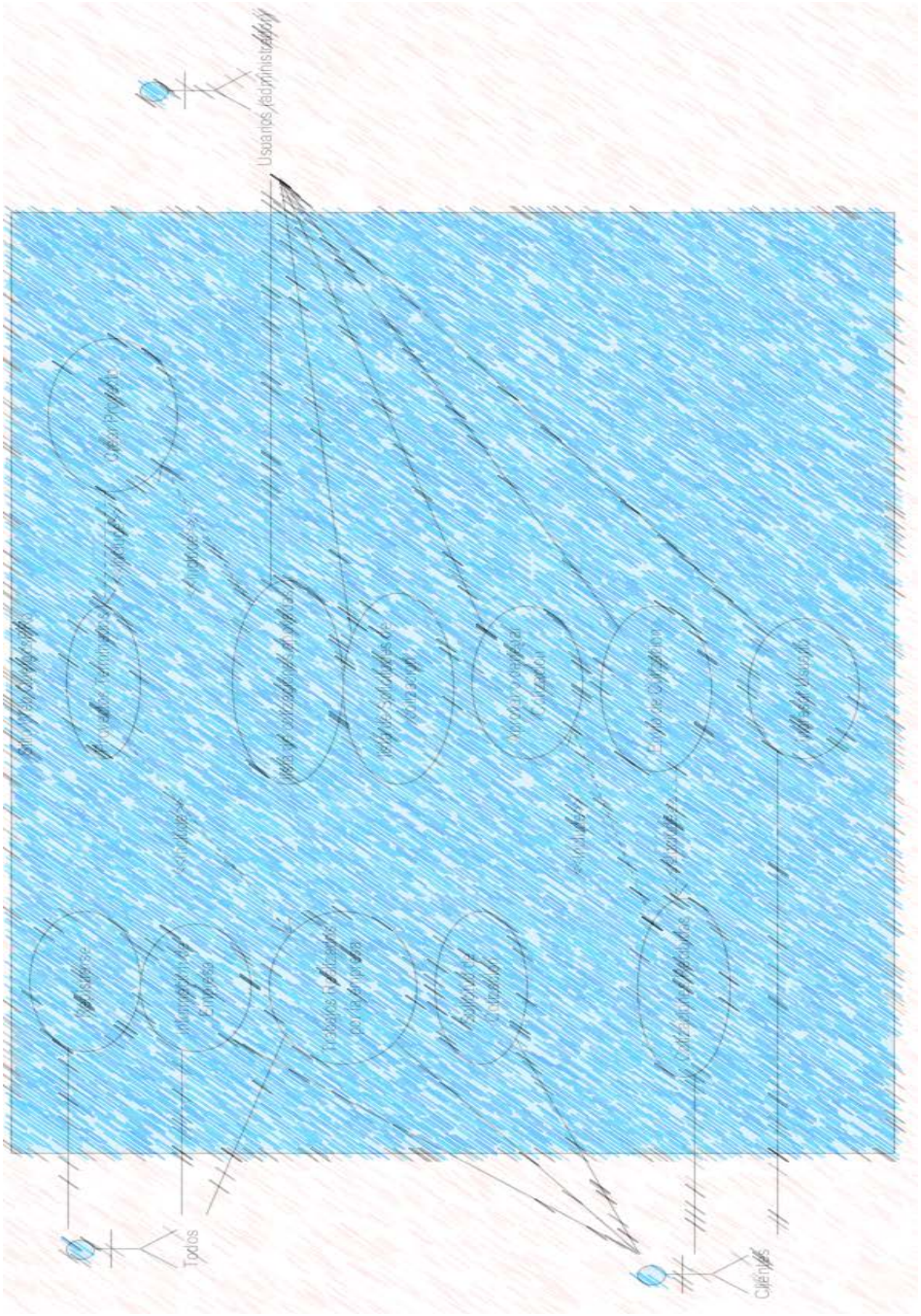
DIAGRAMA DE CONTEXTO



# DIAGRAMA CERO



# DIAGRAMA DE CASO DE USO

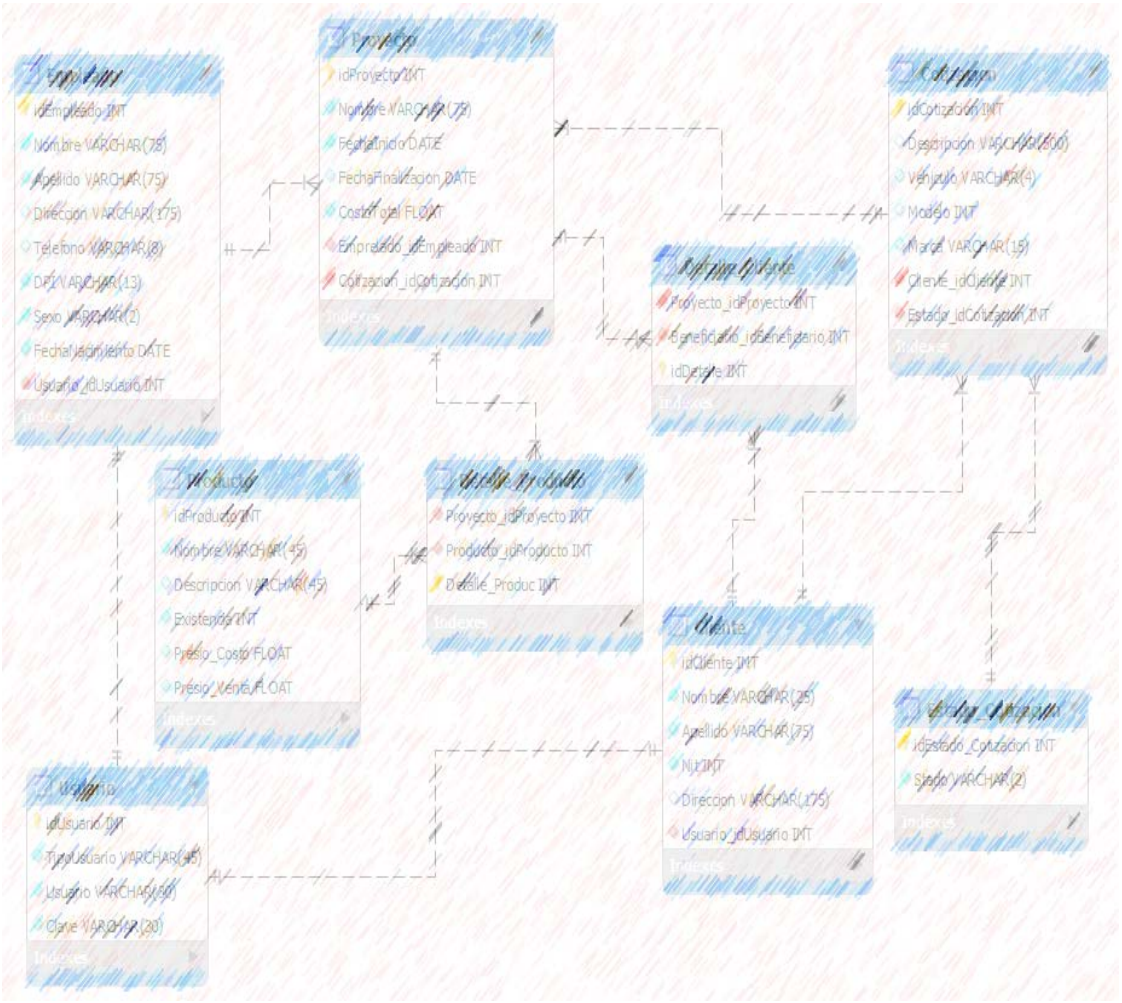


# DIAGRAMA DE ESTADOS

# DIAGRAMA DE SECUENCIA

# DIAGRAMA DE CLASES

# MODELO ENTIDAD RELACIÓN



# DIAGRAMA RELACIONAL

# DOCUMENTACIÓN API

# CONCLUSIONES

# RECOMENDACIONES

# BIBLIOGRAFÍA


- 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/

# GLOSARIO