

Universidad de Pamplona  
Facultad de Ingenierías y Arquitectura  
Programa de Ingeniería de Sistemas

Trabajo de grado presentado para optar al título de ingeniero de sistemas

Tema:  
Estrategia de integración de diferentes plataformas web dinámicas mediante un  
framework del lado del cliente

Autor:  
Fabian Fernney Mendoza Bautista

Pamplona, Norte de Santander  
Febrero de 2019

Universidad de Pamplona  
Facultad de Ingenierías y Arquitectura  
Programa de Ingeniería de Sistemas

Trabajo de grado presentado para optar al título de ingeniero de sistemas

Tema:

Estrategia de integración de diferentes plataformas web dinámicas mediante un framework del lado del cliente

Autor:

Fabian Fernney Mendoza Bautista

Director:

Edgar Alexis Albornoz Espinel  
Mg. En Ciencias de la Computación

Pamplona, Norte de Santander  
Febrero de 2019

## **CITAS**

**“Divide y vencerás”**

**Julio César**

## DEDICATORIA

*A mis padres que siempre me han apoyado para lograr mis metas.*

*A mi hermana que siempre estuvo pendiente de mi formación profesional apoyándome en los momentos difíciles y buenos en el transcurso de mi vida.*

*A mi hija y novia que siempre me apoyaron en el logro de mis metas.*

*A todas de las personas y profesionales que aportaron a mi crecimiento personal, profesional y emocional durante mi formación.*

*A Dios que siempre me ilumina y guía mi camino para ser cada día mejor.*

## Resumen

El desarrollo de aplicaciones Web, buscan maximizar el rendimiento de los servicios que se prestan y que este sea transparente para el usuario final, en la búsqueda de mejorar este rendimiento se llevó a cabo un estudio de las diferentes tecnologías web existentes enfocándolas desde el frontend y el backend, el cual se formalizó para determinar un análisis, clasificación y selección de las herramientas con mejor rendimiento que se utilizan hoy en día en el desarrollo de software web, se complementó con un estudio de los diferentes modelos que proponen varios autores para la integración de plataformas web, este se hace con el fin de analizar diferentes tipos de vistas que se proponen para integrar tecnologías web, con base a este análisis se logró crear una estrategia que permite integrar diferentes plataformas web en una sola aplicación, para validar la estrategia se implementó un prototipo el cual fue desarrollado con diferentes funcionalidades que se encuentren distribuidas en distintos microservicios y desarrolladas con diferentes tecnologías, las cuales fueron construidas bajo la arquitectura de desarrollo RESTful del lado del servidor y AngularJS del lado del cliente, con el propósito de evaluar su eficiencia, uso y comportamiento.

## **Absract**

The development of Web applications, they seek to maximize the performance of the services that lend and that this one is transparent for the final user, in the search of improving this performance there was carried out a study of the different web existing technologies focusing them from the frontend and the backend, which was formalized to determine an analysis, classification and selection of the tools with better performance that they use nowadays in the development of web software, complemented itself with a study of the different models that several authors propose for the integration of web platforms, this one is done in order to analyze different types of conference that they propose to integrate web technologies, with base to this analysis it was achieved to create a strategy that allows to integrate different web platforms in an alone application, To validate the strategy there was implemented a prototype which was developed by different functionalities that are distributed in different microservices and developed with different technologies, which were constructed under the architecture of development RESTful of the side of the servant and AngularJS of the side of the client, with the intention of evaluating his efficiency, use and behavior.

# Tabla de contenido

1	Introducción .....	1
1.1	Planteamiento del problema.....	1
1.2	Justificación .....	2
1.3	Delimitación .....	2
2	Conceptos preliminares .....	4
2.1	Conceptos generales de frameworks Frontend .....	4
2.2	Conceptos generales de Backend .....	10
2.3	Servicio Web .....	14
2.4	Json.....	27
2.5	XML.....	28
2.6	Microservicio.....	28
2.7	SPA.....	30
2.8	Web dinámica.....	33
2.9	Protocolo HTTP.....	33
3	Estrategias de acceso a plataformas web dinámicas .....	34
3.1	Estado del arte .....	34
3.2	Modelos propuestos de la arquitectura de microservicios .....	43
4	Diseño de la estrategia de integración de plataformas web dinámicas. ....	53
4.1	Aspectos técnicos .....	60
4.2	Características, ventajas y desventajas. ....	62
4.3	Validación del prototipo integrasoft .....	63
4.3.1	Descripción del prototipo integrasoft.....	64
4.3.2	Funcionamiento interno del prototipo.....	65
4.3.3	Funcionalidades implementadas en el prototipo. ....	66
4.3.3.1	Login.....	66
4.3.3.2	Gestión de usuarios .....	73
4.4	Análisis de validación de la estrategia .....	83
5	Conclusiones .....	85
6	Recomendaciones y Trabajos Futuros .....	86
7	Bibliografía.....	87

## Índice de tablas

Tabla 1. Comparacion de framework del lado del cliente. ....	9
Tabla 2. Comparación de tecnologías del lado del servidor. ....	14
Tabla 3. Comandos básicos de REST con SQL. ....	19
Tabla 4. Comandos básicos de REST. ....	22
Tabla 5. Ejemplo de uso de REST. ....	22
Tabla 6. Códigos de estados de peticiones HTTP. ....	23
Tabla 7. Comparación de REST vs SOAP. ....	26
Tabla 8. Diferencias entre REST y SOAP. ....	27
Tabla 9. Análisis de las diferentes tecnologías backend y frontend .....	42
Tabla 10. Análisis de los modelos propuestos para el uso de microservicios. ..	52

## Índice de figuras

Figura 1. Ejemplo básico estructura XML. ....	21
Figura 2. Representación básica de un archivo XML. ....	21
Figura 3. Esquema de uso de los microservicios. ....	30
Figura 4. Características de SPA vs Web Tradicional. ....	31
Figura 5. Arquitectura de SPA vs la Web Tradicional. ....	32
Figura 6. Puerta de enlace API para comunicar cliente-servidor. ....	44
Figura 7. Variación: backend para frontend. ....	45
Figura 8. Modelo de integración de ventajas tecnológicas. ....	46
Figura 9. Prototipo de integración de tecnologías. ....	47
Figura 10. Esquema de una arquitectura lógica de microservicios. ....	49
Figura 11. Esquema del funcionamiento interno de las peticiones. ....	50
Figura 12. Funcionalidades por plataforma web en una sola aplicación web. ....	54
Figura 13. Comunicación de usuarios con funcionalidades. ....	54
Figura 14. Flujo básico entre el frontend y el backend. ....	55
Figura 15. Estructura de una aplicación web SPA. ....	56
Figura 16. Estructura de la aplicación web por acceso. ....	57
Figura 17. Acceso del cliente a las funcionalidades. ....	58
Figura 18. Estructura de acceso a la aplicación web por usuario. ....	59
Figura 19. Modelo entidad – relación (MER) de la base de datos. ....	65
Figura 20. Storyboard de funcionalidad login. ....	68
Figura 21. Diagrama de secuencia de la funcionalidad login. ....	69
Figura 22. Método para ingresar a la aplicación web. ....	70
Figura 23. API de la funcionalidad login. ....	71
Figura 24. Métodos de acceso a los datos (DAO). ....	72
Figura 25. Storyboard buscar usuario. ....	73

Figura 26. Storyboard crear usuario. ....	74
Figura 27. Storyboard actualizar usuario. ....	74
Figura 28. Storyboard eliminar usuario. ....	75
Figura 29. Storyboard asignar roles. ....	75
Figura 30. Diagrama de secuencia de la funcionalidad gestión de usuarios. ....	76
Figura 31. Método para buscar usuario desde el frontend.....	77
Figura 32. Método para buscar usuario del API gestión de usuarios. ....	77
Figura 33. Método de acceso a los datos (DAO).....	78
Figura 34. Método para registrar usuario desde el frontend.....	79
Figura 35. Método para registrar usuario del API gestión de usuarios. ....	79
Figura 36. Método de acceso a los datos (DAO).....	80
Figura 37. Método para actualizar usuario desde el frontend.....	81
Figura 38. Método para actualizar usuario del API gestión de usuarios. ....	81
Figura 39. Método de acceso a los datos (DAO).....	82
Figura 40. Método para eliminar usuario desde el frontend.....	82
Figura 41. Método para eliminar usuario del API gestión de usuarios. ....	83
Figura 42. Método de acceso a los datos (DAO).....	83

# 1 Introducción

El avance de la tecnología web que se usa hoy en día ha cambiado drásticamente en los últimos años, esto se debe a la gran variedad de aportes tecnológicos, entre ellos tenemos los frameworks web que facilitan muchos de los trabajos de desarrollo web y ahorro de tiempo en cuanto a la elaboración de código fuente para dichas aplicaciones, de igual forma los avances referentes a los servicios web han traído grandes beneficios, ya que esta tecnología utiliza protocolos que sirven para intercambiar datos entre aplicaciones.

En el presente trabajo se definió una estrategia de integración de plataformas web dinámicas que facilitan la comunicación entre el frontend y el backend, ya que cada una de ellas va a operar de forma independiente y en diferentes máquinas con la implementación de diversas plataformas de programación web.

El objeto de estudio se centró en buscar una estrategia para integrar diferentes plataformas web dinámicas (PHP, JAVA, Python, etc..) en una sola aplicación web, esto se debe al fin de software que existen en la actualidad y en ocasiones presentan fallos en congestión, seguridad, integridad, entre otros aspectos, por lo que su interfaz de usuario y código de acceso a datos se combinan en uno solo (aplicación web monolítica), debido a este factor se busca separar el cliente con el servidor, con el objetivo de minimizar estos fallos.

Con base en esto, se logró construir un prototipo que puede integrar varias tecnologías web, funcionando como una sola aplicación sin que el cliente se dé cuenta que tecnología está invocando, esto se alcanzó con la implementación de los microservicios que operan de forma independiente en distintas máquinas del lado del backend, logrando manipular diferentes tecnologías en una sola del lado del frontend.

## 1.1 Planteamiento del problema

En los procesos de aplicaciones web, los primeros pasos consisten en seleccionar una sola tecnología y una vez tomada esta decisión todo el proyecto es desarrollado bajo la tecnología seleccionada. Esto trae grandes beneficios para un grupo determinado de desarrolladores, quienes solo tienen que implementar un solo estilo de desarrollo en su proyecto.

Sin embargo, esto puede ser un problema si la aplicación tiene una demanda grande de usuarios. Cuando ellos se conecten masivamente realizan peticiones al servidor y este puede colapsar, por capacidad de procesamiento o almacenamiento en memoria (Bianca Schroeder, 2002).

Para resolver este problema, se propone la implementación de una estrategia que permita la integración de tecnologías web existentes JSP, PHP, Python, entre otras, que ofrezcan servicios web con REST. Ello permite dividir las funcionalidades en distintas máquinas, minimizando recursos de hardware y logrando un acceso óptimo para el usuario.

## 1.2 Justificación

Definir una estrategia que permita la integración de distintas tecnologías – **backend** – del lado del servidor, que resuelva problemas de congestión y consumo excesivo de recursos del hardware, esto es importante para que una sola aplicación distribuya su procesamiento por funcionalidades en distintas máquinas, sin que el usuario de la aplicación sea consciente de este hecho.

Para ello, se proyecta utilizar plataformas web que ofrezcan la solicitud de recursos, que serán accedidas mediante métodos HTTP utilizando diferentes tecnologías ya sea JSP, PHP u otras localizadas en diferentes máquinas, que puedan enviar y/o traer datos en distintos formatos de texto ligero como XML o JSON, y poder ser accedidas con la utilización de un framework del lado del cliente – **frontend** – y que permita a los servidores web ofrecer un servicio ágil y rápido al usuario final, esto se verá reflejado cuando el servidor este congestionado de diversas peticiones a la vez, también ayudará a disminuir el tiempo total de respuesta de una invocación al servicio web, y brindará un gran aporte a la hora de desarrollar aplicaciones para las diferentes compañías, ya que ofrecerá una mayor visibilidad, fiabilidad y escalabilidad para convertirse en productos más flexibles, para tener una gran libertad al momento de cambiar o probar nuevos entornos dentro del desarrollo web.

## 1.3 Delimitación

### Objetivo general

Definir una estrategia de integración de diferentes plataformas web dinámicas del lado del servidor mediante un framework del lado del cliente.

## **Objetivos específicos**

- Establecer un estudio sobre las diferentes tecnologías web dinámicas para el frontend y el backend.
- Diseñar una estrategia que permita el acceso a las diferentes plataformas web dinámicas a través de un framework del lado del cliente.
- Validar la estrategia de la integración de las diferentes plataformas web dinámicas a través de un framework del lado del cliente, mediante un desarrollo backend y frontend, para determinar si se ajusta a las plataformas de desarrollo web.

## **Acotaciones**

La estrategia no incluye:

- Tolerancia a fallos.
- Fallos de seguridad.
- Clúster de bases de datos distribuidos.
- Web Services.
- Más de cuatro funcionalidades durante el desarrollo del prototipo.

## 2 Conceptos preliminares

El presente capítulo se enfocó en determinar los diversos conceptos que se manejan en el desarrollo de software web y el manejo de las tecnologías web existentes más utilizadas por los desarrolladores, como herramientas para solucionar numerosos problemas en la construcción de software web, a su vez se realizaron comparativos entre las tecnologías mencionadas a continuación, para poder analizar y dar una breve conclusión de su importancia y uso en su aplicación, en el cual se documentan las tecnologías más adecuadas en el desarrollo tanto del frontend (lado del cliente) como del backend (lado del servidor) para seleccionar las tecnologías más apropiadas para la construcción de aplicaciones web dinámicas.

### 2.1 Conceptos generales de frameworks Frontend

Frontend es la parte de un programa o dispositivo a un usuario puede acceder directamente. Son todas las tecnologías de diseño y desarrollo web que corren en el navegador y que se encargan de la interactividad con los usuarios. (Nicole Chapaval, 2018)

HTML, CSS y JavaScript son los lenguajes principales del Frontend, de los que se desprenden una cantidad de frameworks y librerías que expanden sus capacidades para crear cualquier tipo de interfaces de usuarios. React, Redux, Angular, Bootstrap, Foundation, LESS, Sass, Stylus y PostCSS son algunos de ellos. (Nicole Chapaval, 2018)

- **ANGULARJS** (comúnmente llamado Angular.js o AngularJS 1), es un framework de JavaScript de código abierto, mantenido por Google, que se utiliza para crear y mantener aplicaciones web de una sola página. Su objetivo es aumentar las aplicaciones basadas en navegador con capacidad de Modelo Vista Controlador (MVC), en un esfuerzo para hacer que el desarrollo y las pruebas sean más fáciles. (Pablo Lázaro, 2013)

La biblioteca lee el HTML que contiene atributos de las etiquetas personalizadas adicionales, entonces obedece a las directivas de los atributos personalizados, y une las piezas de entrada o salida de la página a un modelo representado por las variables estándar de JavaScript. Los valores de las variables de JavaScript se pueden configurar manualmente,

o recuperados de los recursos JSON estáticos o dinámicos. (Pablo Lázaro, 2013)

- **REACT** es una biblioteca escrita en JavaScript, desarrollada en Facebook para facilitar la creación de componentes interactivos, reutilizables, para interfaces de usuario. Se utiliza en Facebook para la producción de componentes, e Instagram está escrito enteramente en React. Uno de sus puntos más destacados, es que no sólo se utiliza en el lado del cliente, sino que también se puede representar en el servidor, y trabajar juntos.

React intenta ayudar a los desarrolladores a construir aplicaciones que usan datos que cambian todo el tiempo. Su objetivo es ser sencillo, declarativo y fácil de combinar. React sólo maneja la interfaz de usuario en una aplicación; está construida únicamente para utilizar el patrón de diseño modelo–vista–controlador (MVC), y puede ser utilizada conjuntamente con otras bibliotecas de Javascript o más grandes #MVC como AngularJS. También puede ser utilizado con las extensiones de React-based que se encargan de las partes no-UI (no gráficas) de una aplicación web. (Julio Grados, 2016)

React.js está construido en torno a hacer funciones, que toman las actualizaciones de estado de la página y que se traduzcan en una representación virtual de la página resultante. Siempre que React es informado de un cambio de estado, vuelve a ejecutar esas funciones para determinar una nueva representación virtual de la página, a continuación, se traduce automáticamente ese resultado en los cambios del DOM necesarios para reflejar la nueva presentación de la página. (Julio Grados, 2016)

A primera vista, esto suena como que fuera más lento que el enfoque JavaScript habitual de actualización de cada elemento, según sea necesario. Detrás de escena, sin embargo, React.js hace justamente eso: tiene un algoritmo muy eficiente para determinar las diferencias entre la representación virtual de la página actual y la nueva. A partir de esas diferencias, hace el conjunto mínimo de cambios necesarios en el DOM. (Julio Grados, 2016)

Pues utiliza un concepto llamado el DOM virtual que hace selectivamente sub-árboles de los nodos sobre la base de cambios de estado, desarrollando esto, con la menor cantidad de manipulación DOM posible, con el fin de mantener los componentes actualizados, estructurando sus datos. (Julio Grados, 2016)

- **EMBER** es un framework para javascript, que nos permite utilizar el patrón MVC (Modelo Vista Controlador) en nuestras aplicaciones de manera más automatizada y sencilla. (Rafael Macías, 2013)

Así como un paquete de iniciación donde encontraremos a parte de esta, otras librerías en las que se apoya Ember. Se apoya en jQuery, una librería javascript, que nos permite manejar nuestra aplicación en el entorno cliente, con toda la potencia de javascript pero de manera más sencilla. También se apoya en Handlebars, una librería javascript que nos permite crear plantillas en el HTML, algo indispensable en el patrón MVC. (Rafael Macías, 2013)

- **VUE** (pronunciado / vju: /, como ver) es un framework progresivo para construir interfaces de usuario. A diferencia de otros marcos monolíticos, Vue está diseñado desde cero para ser adoptable incrementalmente. La biblioteca central está enfocada solo en la capa de visualización, y es fácil de seleccionar e integrar con otras bibliotecas o proyectos existentes. Por otro lado, Vue también es perfectamente capaz de impulsar sofisticadas aplicaciones de una sola página cuando se utiliza en combinación con herramientas modernas y bibliotecas de soporte. (Evan You, 2014)

Fue creado por Evan You que trabajaba en Google realizando prototipos y en el core del Framework de Meteor, hasta que pensó en otra forma de hacer una opción más fácil que abarcara las necesidades a la hora de hacer prototipos. Así surgió Vue en el 2014, desde entonces ha tenido una gran evolución y sigue creciendo en su versión 2 cada vez más y más. (Evan You, 2014)

- **BACKBONE** es una librería de JavaScript basada en el patrón de arquitectura MV\* que trabaja del lado del cliente (Front-end), que nos ayuda a estructurar de una manera más eficaz nuestras aplicaciones web. Puede trabajar con cualquier tecnología del lado del Servidor (Backend) ya sea Ruby on Rails, Laravel, Django, etc. (Jesus Rosas, 2015)
  - Es basado en el patrón de arquitectura MVC utilizando una variante llamada MV\*(Rutas y Colecciones).
  - No sigue un paradigma estricto como lo es Angular.js.
  - Su sistema de templates viene por separado por lo que es más fácil realizar algunas modificaciones en el core (Underscore.js).
  - Se integra de manera fácil con cualquier framework del lado del Servidor

- Promueve las buenas prácticas de desarrollo

Así mismo Backbone.js da una mejor estructura a las aplicaciones web al ofrecer modelos con unión clave-valor y eventos personalizados, así como una gran cantidad de colecciones con una rica API de funciones enumerables, vistas con eventos declarativos y todo esto conectado a su API existente sobre una interfaz RESTful. (Jesus Rosas, 2015)

## COMPARATIVA ENTRE LAS TECNOLOGÍAS FRONTEND

ÍTEM	TIPO	OPEN SOURCE	VENTAJAS	DESVENTAJAS
<b>ANGULARJS</b>	Framework	Si	<ul style="list-style-type: none"> <li>• Construido por Google.</li> <li>• Framework MVC del lado del cliente.</li> <li>• Intuitivo.</li> <li>• Exhaustivo.</li> <li>• Proporciona SAP de una manera muy limpia y sostenible.</li> <li>• Proporciona capacidad de enlace de datos a HTML, lo que brinda al usuario una experiencia rica y receptiva</li> <li>• Usa inyección de dependencia y hace uso de la separación de preocupaciones.</li> <li>• Proporciona componentes reutilizables.</li> <li>• Se escribe menos código y obtiene más funcionalidad.</li> <li>• Las vistas son páginas HTML puras, y los controladores escritos en JavaScript hacen el procesamiento del negocio.</li> <li>• En constante desarrollo y de código abierto</li> </ul>	<ul style="list-style-type: none"> <li>• JavaScript Obligatorio</li> <li>• Grande y complicado.</li> <li>• Más de 2000 observadores pueden retrasar severamente la IU.</li> <li>• Hay que utilizar las nuevas versiones.</li> <li>• Las implementaciones angulares escalan mal.</li> <li>• No seguro.</li> <li>• La autenticación y autorización del lado del servidor es imprescindible para mantener una aplicación segura.</li> <li>• Si se deshabilita JavaScript del lado del navegador la aplicación no funciona.</li> </ul>
<b>REACT</b>	Librería	Si	<ul style="list-style-type: none"> <li>• El DOM virtual en ReactJS mejora la experiencia del usuario.</li> <li>• El trabajo del desarrollador es más rápido</li> </ul>	<ul style="list-style-type: none"> <li>• Alto ritmo de desarrollo.</li> <li>• Mala documentación.</li> <li>• La orientación a la vista es uno de los contras de ReactJS.</li> </ul>

			<ul style="list-style-type: none"> <li>• Reutilizar los componentes React ahorra tiempo significativamente.</li> <li>• El flujo de datos en una dirección en ReactJS proporciona un código estable</li> <li>• En constante desarrollo y de código abierto.</li> </ul>	<ul style="list-style-type: none"> <li>• Se debe encontrar modelo y controlador para resolver el problema vistas.</li> <li>• Biblioteca de gran tamaño de React.</li> <li>• React.js es solo una capa de vista.</li> <li>• Complejidad cuando se empieza a utilizar.</li> </ul>
<b>EMBER</b>	Framework	Si	<ul style="list-style-type: none"> <li>• Alto rendimiento.</li> <li>• Desarrollo más rápido debido a Ember CLI.</li> <li>• Documentación comprensible.</li> <li>• Enlace de datos bidireccional.</li> <li>• Bien organizado.</li> <li>• Herramienta propia de depuración (Ember Inspector).</li> </ul>	<ul style="list-style-type: none"> <li>• Complicaciones con el procesamiento de cambios rápidos.</li> <li>• El más pesado de los frameworks.</li> <li>• Comunidad muy pequeña.</li> <li>• Difícil de aprender.</li> <li>• Demasiado grande para proyectos pequeños.</li> <li>• Lanza lentamente nuevas versiones con pocas opciones nuevas.</li> </ul>
<b>VUE</b>	Framework	Si	<ul style="list-style-type: none"> <li>• Fácil de aprender y fácil de usar.</li> <li>• Ligereza.</li> <li>• Documentación bien escrita.</li> <li>• Uso estrecho (crea interfaces de usuario para la web).</li> <li>• Aplicable tanto para aplicaciones simples como complejas.</li> <li>• Se puede usar para desarrollar y una aplicación desde cero o para agregar nuevas características a la existente.</li> <li>• En constante desarrollo y de código abierto.</li> </ul>	<ul style="list-style-type: none"> <li>• Comunidad relativamente pequeña.</li> <li>• Monopolio en el desarrollo del proyecto.</li> <li>• Difícil de reconocer debido a la flexibilidad.</li> <li>• Muchos de los complementos están en chino.</li> <li>• Errores de tiempo de ejecución no descriptivos en las plantillas.</li> <li>• Falta de componentes estables.</li> </ul>
<b>BACKBONE</b>	Librería	Si	<ul style="list-style-type: none"> <li>• La biblioteca tiene un código fuente abierto anotado. Esto significa que cualquier desarrollador puede estudiarlo y usarlo para el desarrollo.</li> <li>• Backbone.js pesa 7.6 KB con gzip (Backbone + JQuery + Underscore = 41.6 KB). Es decir, puede</li> </ul>	<ul style="list-style-type: none"> <li>• La libre elección de herramientas para extender la aplicación construida sobre la base de Backbone.js puede presentar un serio desafío para los principiantes. Por lo tanto, antes de comenzar el trabajo, se recomienda pensar en las bibliotecas que</li> </ul>

			<p>referirse a los juegos de herramientas livianas.</p> <ul style="list-style-type: none"> <li>• Una biblioteca muy compacta: el conjunto minimalista más simple de clases y funciones que cubre todos los componentes más importantes para construir una aplicación web.</li> <li>• Muy flexible y bien combinado con otras herramientas. La elección de las herramientas y complementos que se utilizarán sigue siendo para el desarrollador.</li> </ul>	<p>se utilizarán y estudiar sus posibilidades.</p> <ul style="list-style-type: none"> <li>• A medida que la aplicación se extiende, las vistas pueden volverse pesadas en términos de código. En algunos casos, este problema se puede eliminar dividiendo una vista en varias de acuerdo con la lógica.</li> </ul>
--	--	--	--	---

*Tabla 1. Comparación de framework del lado del cliente.  
Fuente: Autor del proyecto.*

**Apreciación:** en la tabla 1 se realizó un análisis de algunos frameworks o librerías que más se utilizan hoy en día para el desarrollo frontend, esto permitió analizar las diferentes ventajas y desventajas que se muestran anteriormente de los frameworks más populares que existen actualmente, retomando como base las ventajas que tienen los frameworks, se sugiere que deben cumplir con las siguientes características; constante desarrollo y código abierto, framework MVC, buena documentación, fácil de usar, trabajo de desarrollo rápido, construcción de aplicaciones web robustas, por esta razón se pueden determinar los beneficios para hacer aplicaciones web rápidas y robustas del lado del cliente, en cuanto a las desventajas se deben evitar frameworks que a medida que crecen la aplicación se vuelva lenta, difícil de aprender, procesamiento de cambios lentos cuando se compila nuevos procesos.

En este sentido, los framework que se destacan son: **AngularJS** el cual es desarrollado por Google, de código abierto y se utiliza en un sinnúmero de plataformas web o móviles que actualmente consumimos, también se resalta **React** como una de las librerías que más usan los desarrolladores por su código estable y fácil de utilizar para la construcción de aplicaciones robustas, y por último **VUE** trae consigo un aprendizaje rápido y fácil de entender a la hora de construir aplicaciones simples y robustas.

## 2.2 Conceptos generales de Backend

Backend es la capa de acceso a datos de un software o cualquier dispositivo, que no es directamente accesible por los usuarios, además contiene la lógica de la aplicación que maneja dichos datos. El Backend también accede al servidor, que es una aplicación especializada que entiende la forma como el navegador solicita cosas. (Nicole Chapaval, 2018)

Algunos de los lenguajes de programación de Backend son Python, PHP, Ruby, C# y Java, y así como en Frontend, cada uno de los anteriores tiene diferentes frameworks que te permiten trabajar mejor según el proyecto que se vaya desarrollando. (Nicole Chapaval, 2018)

- **Python** es un lenguaje multiparadigma, esto significa que combina propiedades de diferentes paradigmas de programación. Principalmente es un lenguaje orientado a objetos, todo en Python es un objeto, pero también incorpora aspectos de la programación imperativa, funcional, procedural y reflexiva. (Manuel Zaforas, 2017)

Una de las características más reseñables de Python es que es un lenguaje interpretado, esto significa que no se compila a diferencia de otros lenguajes como Java o C/C++, sino que es interpretado en tiempo de ejecución. Además, es de tipado dinámico, aunque opcionalmente desde la versión 3.5 podemos hacer uso de tipado estático. (Manuel Zaforas, 2017)

Python es cross plataforma, es decir, podemos ejecutarlo en diferentes sistemas operativos como Windows o Linux simplemente usando el intérprete correspondiente. (Manuel Zaforas, 2017)

Algunos le achacan a Python que es más lento en tiempo de ejecución que otros lenguajes compilados como Java o C/C++. Y es cierto, al tratarse de un lenguaje interpretado, Python es más lento. (Manuel Zaforas, 2017)

- **PHP** (acrónimo recursivo de PHP: Hypertext Preprocessor) es un lenguaje de código abierto muy popular especialmente adecuado para el desarrollo web y que puede ser incrustado en HTML. (The PHP Group, 2018)

En lugar de usar muchos comandos para mostrar HTML (como en C o en Perl), las páginas de PHP contienen HTML con código incrustado que hace "algo" (en este caso, mostrar "¡Hola, soy un script de PHP!"). El código de PHP está encerrado entre las etiquetas especiales de comienzo y final

<?php y ?> que permiten entrar y salir del "modo PHP". (The PHP Group, 2018)

Lo que distingue a PHP de algo del lado del cliente como Javascript es que el código es ejecutado en el servidor, generando HTML y enviándolo al cliente. El cliente recibirá el resultado de ejecutar el script, aunque no se sabrá el código subyacente que era. El servidor web puede ser configurado incluso para que procese todos los ficheros HTML con PHP, por lo que no hay manera de que los usuarios puedan saber qué se tiene debajo de la manga. (The PHP Group, 2018)

Lo mejor de utilizar PHP es su extrema simplicidad para el principiante, pero a su vez ofrece muchas características avanzadas para los programadores profesionales. No sienta miedo de leer la larga lista de características de PHP. En unas pocas horas podrá empezar a escribir sus primeros scripts. (The PHP Group, 2018)

Aunque el desarrollo de PHP está centrado en la programación de scripts del lado del servidor, se puede utilizar para muchas otras cosas. (The PHP Group, 2018)

- **Java** es un lenguaje de programación creado en 1995 para el entorno de computación de mismo nombre por Sun Microsystems. Java tiene un conjunto de reglas que determinan cómo se escriben las instrucciones. Estas reglas se conocen como su sintaxis. Una vez que se ha escrito un programa, las instrucciones de alto nivel se traducen en códigos numéricos que las computadoras pueden entender y ejecutar. (Vangie Beal, 2018)

Su filosofía WORA (Write Once, Run Anywhere) permite a los desarrolladores portar sus aplicaciones a distintos sistemas sin apenas -o ningún- esfuerzo, aunque la gran variedad de dispositivos en los que se puede ejecutar Java lleva la segmentación entre applets Java, aplicaciones de escritorio, aplicaciones empresariales y aplicaciones para teléfonos móviles. (Vangie Beal, 2018)

Java se define como un lenguaje orientado a objetos similar a C ++, pero se simplifica para eliminar las características del lenguaje que causan errores comunes de programación. Los códigos fuente de los archivos (archivos con un .java extensión) se compilan en un formato llamado código de bytes (archivos con un .class extensión), que luego puede ser ejecutado por un Java intérprete. El código Java compilado se puede ejecutar en la mayoría de las computadoras porque existen intérpretes de Java y entornos de

tiempo de ejecución, conocidos como Máquinas Virtuales Java (VM), para la mayoría de los sistemas operativos, incluido UNIX, elMacintosh OS, y Windows. (Vangie Beal, 2018)

- **Ruby** es un código abierto, interpretado, orientado a objetos lenguaje de programación creado por Yukihiro Matsumoto, que eligió el nombre de la piedra preciosa para sugerir "una joya de un idioma." Ruby está diseñado para ser simple, completo, extensible y portátil. Desarrollado principalmente en Linux, Ruby funciona en la mayoría de las plataformas, como la mayoría de las plataformas basadas en UNIX, DOS, Windows, Macintosh, BeOS y OS/2, por ejemplo. Según los defensores, la sintaxis simple de Ruby (parcialmente inspirada por Ada y Eiffel), lo hace legible por cualquiera que esté familiarizado con cualquier lenguaje de programación moderno. (Margaret Rouse, 2016)

Ruby se considera similar a Smalltalk y Perl. Los autores del libro Programming Ruby: The Pragmatic Programmer's Guide, David Thomas y Andrew Hunt dicen que está completamente orientado a objetos, como Smalltalk, aunque es más convencional de usar, y tan conveniente como Perl, pero totalmente orientado a objetos, que conduce a programas mejor estructurados y fáciles de mantener. Para cumplir con los principios de programación extrema (XP), Ruby permite que porciones de proyectos se escriban en otros idiomas si son más adecuados. (Margaret Rouse, 2016)

- **C #** es un lenguaje elegante y seguro orientado a objetos que permite a los desarrolladores crear una variedad de aplicaciones seguras y robustas que se ejecutan en .NET Framework. Puede usar C # para crear aplicaciones cliente de Windows, servicios web XML, componentes distribuidos, aplicaciones cliente-servidor, aplicaciones de bases de datos y mucho, mucho más. Visual C # proporciona un editor de código avanzado, diseñadores de interfaz de usuario convenientes, depurador integrado y muchas otras herramientas para facilitar el desarrollo de aplicaciones basadas en el lenguaje C # y .NET Framework. (Microsoft, 2015)

**Lenguaje C #:** la sintaxis de C # es altamente expresiva, pero también es simple y fácil de aprender. La sintaxis de llavero de C # será instantáneamente reconocible para cualquiera que esté familiarizado con C, C ++ o Java. Los desarrolladores que conocen cualquiera de estos lenguajes normalmente pueden comenzar a trabajar productivamente en C # en muy poco tiempo. La sintaxis de C # simplifica muchas de las complejidades de C ++ y proporciona características potentes como tipos de valores anulables, enumeraciones, delegados, expresiones lambda y

acceso directo a la memoria, que no se encuentran en Java. C # admite métodos y tipos genéricos, que proporcionan mayor seguridad y rendimiento de tipo, e iteradores, que permiten a los implementadores de clases de recopilación definir comportamientos de iteración personalizados que son fáciles de usar por el código del cliente. Las expresiones Language-Integrated Query (LINQ) hacen de la consulta fuertemente tipada una construcción de lenguaje de primera clase. (Microsoft, 2015)

## COMPARATIVA ENTRE LAS TECNOLOGÍAS BACKEND

ÍTEM	OPEN SOURCE	VENTAJAS	DESVENTAJAS
<b>Python</b>	<b>Si</b>	<ul style="list-style-type: none"> <li>• Extensas bibliotecas de soporte.</li> <li>• Característica de integración.</li> <li>• Productividad mejorada del programador.</li> </ul>	<ul style="list-style-type: none"> <li>• Velocidad.</li> <li>• Desarrollo móvil.</li> <li>• Consumo de memoria.</li> <li>• Acceso a la base.</li> <li>• Errores en tiempo de ejecución.</li> </ul>
<b>PHP</b>	<b>Si</b>	<ul style="list-style-type: none"> <li>• Fácil de aprender.</li> <li>• Código abierto.</li> <li>• Portabilidad.</li> <li>• Rendimiento rápido.</li> <li>• Comunidad extensa.</li> <li>• No es necesario escribir código adicional</li> <li>• Simplifica el mantenimiento de aplicaciones web</li> <li>• Automatiza tareas comunes de desarrollo web</li> <li>• Buena documentación</li> </ul>	<ul style="list-style-type: none"> <li>• La calidad de los frameworks PHP difiere</li> <li>• Afecta la velocidad y el rendimiento de los sitios web</li> <li>• Seguridad</li> <li>• No apto para grandes aplicaciones</li> </ul>
<b>Ruby</b>	<b>Si</b>	<ul style="list-style-type: none"> <li>• Admite la metaprogramación.</li> <li>• Rápido para escribir.</li> <li>• Todo es un objeto.</li> <li>• Depuración sencilla.</li> <li>• Gran rendimiento de la aplicación.</li> <li>• Productividad.</li> <li>• Arquitectura Modelo-Vista-Controlador (MVC).</li> <li>• Fácil mantenimiento y actualizaciones de código.</li> </ul>	<ul style="list-style-type: none"> <li>• Registro activo</li> <li>• Multithreading</li> <li>• Documentación</li> <li>• La velocidad de arranque.</li> <li>• Procesamiento lento.</li> </ul>
<b>Java</b>	<b>Si</b>	<ul style="list-style-type: none"> <li>• Facilidad de uso.</li> <li>• Confiabilidad.</li> <li>• Seguridad.</li> <li>• Independencia de la plataforma.</li> <li>• Orientada a objetos</li> </ul>	<ul style="list-style-type: none"> <li>• Lenguaje de un solo paradigma</li> <li>• Rendimiento</li> <li>• Lenguaje de latencia limitada</li> <li>• No es compatible con la programación de bajo nivel</li> </ul>

		<ul style="list-style-type: none"> <li>• Distribuido</li> <li>• Asignación</li> <li>• Multiproceso</li> </ul>	<ul style="list-style-type: none"> <li>• Sin control sobre la recolección de basura</li> <li>• Toma más espacio de memoria</li> </ul>
<b>C#</b>	<b>No</b>	<ul style="list-style-type: none"> <li>• La biblioteca de clases .net permitirá el desarrollo rápido de prototipos.</li> <li>• Lenguaje orientado a objetos.</li> <li>• Recolección automática de basura.</li> <li>• No hay problema si la fuga de memoria.</li> <li>• Fácil de desarrollar.</li> <li>• Programación de soporte.</li> </ul>	<ul style="list-style-type: none"> <li>• Lento de ejecutar</li> <li>• Es menos flexible que C ++</li> <li>• El código administrado puede ser más lento que el código nativo.</li> <li>• La migración de aplicaciones a .NET puede ser costosa.</li> </ul>

*Tabla 2. Comparación de tecnologías del lado del servidor.  
Fuente: Autor del proyecto*

**Apreciación:** en la tabla 2 se pudo diferenciar las diferentes características de las tecnologías web del lado del servidor, de acuerdo sus ventajas y desventajas se recomienda hacer uso de tecnologías que tengan las siguientes ventajas: constante desarrollo y de código abierto, facilidad de uso, buen soporte (documentación), orientado a objetos, automatización de tareas de desarrollo, seguridad, multiplataforma y productividad, de igual forma se deben evitar tecnologías web que sea de procesamiento lento, no se pueda utilizar en múltiples plataformas, documentación no entendible, errores en tiempo de ejecución.

En este sentido se sugiere utilizar tecnologías como **JAVA, PHP y Python**, donde se puede apreciar, que estas tecnologías son las más apropiadas y utilizadas hoy en día por las diferentes aplicaciones que utilizamos actualmente, de este modo, esto trae grandes beneficios para los desarrolladores, ya que cada una de ellas cuenta con notables avances para la construcción backend.

## 2.3 Servicio Web

El consorcio W3C define los Servicios Web como sistemas software diseñados para soportar una interacción interoperable máquina 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. (Rafael Navarro, 2006)

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).

Las primeras herramientas para Servicios Web estaban centradas en esta visión. Algunos lo llaman la primera generación de Servicios Web. Esta es la razón por la que este estilo está muy extendido. Sin embargo, ha sido algunas veces criticado por no ser débilmente acoplado, ya que suele ser implementado por medio del mapeo de servicios directamente a funciones específicas del lenguaje o llamadas a métodos. Muchos especialistas creen que este estilo debe desaparecer.

- 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.

Los Servicios Web basados en SOA son soportados por la mayor parte de desarrolladores de software y analistas. Al contrario que los Servicios Web basados en RPC, este estilo es débilmente acoplado, lo cual es preferible ya que se centra en el “contrato” proporcionado por el documento WSDL, más que en los detalles de implementación subyacentes.

- 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 estándar (por ejemplo, GET, PUT, ...). Por tanto, este estilo se centra más en interactuar con recursos con estado, que con mensajes y operaciones.

En este apartado se va profundizar más en la arquitectura que establece REST, referente a SOAP se llevó una comparativa ante REST, según el autor de este artículo:

## **REST**

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. (Rafael Navarro, 2006)

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 dominio 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. (Rafael Navarro, 2006)

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.

## **MOTIVACIÓN DE REST**

La motivación de REST es la de capturar las características de la Web que la han hecho tan exitosa. (Rafael Navarro, 2006)

Si pensamos un poco en este éxito, nos daremos cuenta que la Web ha sido la única aplicación distribuida que ha conseguido ser escalable al tamaño de Internet. El éxito lo debe al uso de formatos de mensaje extensibles y estándares, pero además cabe destacar que posee un esquema de direccionamiento global (estándar y extensible a su vez). (Rafael Navarro, 2006)

En particular, el concepto central de la Web es un espacio de URIs unificado. Las URIs permiten la densa red de enlaces que permiten a la Web que sea tan utilizada. Por tanto, ellos consiguen tejer una mega-aplicación. (Rafael Navarro, 2006)

Las URIs identifican recursos, los cuales son objetos conceptuales. La representación de tales objetos se distribuye por medio de mensajes a través de la Web. Este sistema es extremadamente desacoplado. (Rafael Navarro, 2006)

Estas características son las que han motivado para ser utilizadas como guía para la evolución de la Web. (Rafael Navarro, 2006)

## PRINCIPIOS DE REST

El estilo de arquitectura subyacente a la Web es el modelo REST. Los objetivos de este estilo de arquitectura se listan a continuación:

- **Escalabilidad de la interacción con los componentes.** La Web ha crecido exponencialmente sin degradar su rendimiento. Una prueba de ellos es la variedad de clientes que pueden acceder a través de la Web: estaciones de trabajo, sistemas industriales, dispositivos móviles, ...
- **Generalidad de interfaces.** Gracias al protocolo HTTP, cualquier cliente puede interactuar con cualquier servidor HTTP sin ninguna configuración especial. Esto no es del todo cierto para otras alternativas, como SOAP para los Servicios Web.
- **Puesta en funcionamiento independiente.** Este hecho es una realidad que debe tratarse cuando se trabaja en Internet. Los clientes y servidores pueden ser puestas en funcionamiento durante años. Por tanto, los servidores antiguos deben ser capaces de entenderse con clientes actuales y viceversa. Diseñar un protocolo que permita este tipo de características resulta muy complicado. HTTP permite la extensibilidad mediante el uso de las cabeceras, a través de las URIs, a través de la habilidad para crear nuevos métodos y tipos de contenido.
- **Compatibilidad con componentes intermedios.** Los más populares intermediarios son varios tipos de proxys para Web. Algunos de ellos, las caches, se utilizan para mejorar el rendimiento. Otros permiten reforzar las políticas de seguridad: firewalls. Y, por último, otro tipo importante de intermediarios, Gateway, permiten encapsular sistemas no propiamente Web. Por tanto, la compatibilidad con intermediarios nos permite reducir la latencia de interacción, reforzar la seguridad y encapsular otros sistemas. (Rafael Navarro, 2006)

REST logra satisfacer estos objetivos aplicando cuatro restricciones:

- Identificación de recursos y manipulación de ellos a través de representaciones. Esto se consigue mediante el uso de URIs. HTTP es un protocolo centrado en URIs. Los recursos son los objetos lógicos a los

que se le envían mensajes. Los recursos no pueden ser directamente accedidos o modificados. Más bien se trabaja con representaciones de ellos. Cuando se utiliza un método PUT para enviar información, se coge como una representación de lo que nos gustaría que el estado del recurso fuera. Internamente el estado del recurso puede ser cualquier cosa desde una base de datos relacional a un fichero de texto.

- Mensajes autodescriptivos. REST dicta que los mensajes HTTP deberían ser tan descriptivos como sea posible. Esto hace posible que los intermediarios interpreten los mensajes y ejecuten servicios en nombre del usuario. Uno de los modos que HTTP logra esto es por medio del uso de varios métodos estándares, muchos encabezamientos y un mecanismo de direccionamiento. Por ejemplo, las cachés Web saben que por defecto el comando GET es cacheable (ya que es side-effect-free) en cambio POST no lo es. Además, saben cómo consultar las cabeceras para controlar la caducidad de la información. HTTP es un protocolo sin estado y cuando se utiliza adecuadamente, es posible interpretar cada mensaje sin ningún conocimiento de los mensajes precedentes. Por ejemplo, en vez de lograrse del modo que lo hace el protocolo FTP, HTTP envía esta información en cada mensaje.
- Hipermedia como un mecanismo del estado de la aplicación. El estado actual de una aplicación Web debería ser capturado en uno o más documentos de hipertexto, residiendo tanto en el cliente como en el servidor. El servidor conoce sobre el estado de sus recursos, aunque no intenta seguirles la pista a las sesiones individuales de los clientes. Esta es la misión del navegador, él sabe cómo navegar de recurso a recurso, recogiendo información que el necesita o cambiar el estado que el necesita cambiar.

En la actualidad existen millones de aplicaciones Web que implícitamente heredan estas restricciones de HTTP. Hay una disciplina detrás del diseño de sitios Web escalables que puede ser aprendida de los documentos de arquitectura Web o de varios estándares. Por otra parte, también es verdad que muchos sitios Web comprometen uno más de estos principios, por ejemplo, seguir la pista de los usuarios moviéndose a través de un sitio. Esto es posible dentro de la infraestructura de la Web, pero daña la escalabilidad, volviendo un medio sin conexión en todo lo contrario. (Rafael Navarro, 2006)

Los defensores de REST han creído que estas ideas son tan aplicables a los problemas de integración de aplicaciones como los problemas de integración de hipertexto. Fielding es bastante claro diciendo que REST no es la cura para todo. Algunas de estas características de diseño no serán apropiadas para otras aplicaciones. Sin embargo, aquellos que han decidido adoptar REST como un

modelo de servicio Web sienten que al menos articula una filosofía de diseño con fortaleza, debilidades y áreas de aplicabilidad documentada. (Rafael Navarro, 2006)

## DISEÑO BASADO EN REST

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. (Rafael Navarro, 2006)

La Web consiste del protocolo HTTP, de tipos de contenido, incluyendo HTML y otras tecnologías tales como el Domain Name System (DNS). (Rafael Navarro, 2006)

Por otra parte, HTML puede incluir javascript y applets, los cuales dan soporte al code-on-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. (Rafael Navarro, 2006)

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 tabla 3:

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

*Tabla 3. Comandos básicos de REST con SQL.*

*Fuente: <http://users.dsic.upv.es/~rnavarro/NewWeb/docs/RestVsWebServices.pdf>*

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. (Rafael Navarro, 2006)

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. (Rafael Navarro, 2006)

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. (Rafael Navarro, 2006)

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. (Rafael Navarro, 2006)

## **INTERFAZ BASADA EN REST**

Antes de empezar a pensar en el servicio, debemos responder las siguientes preguntas en orden:

- ¿Qué son las URIs? Las cosas identificadas por URIs son “recursos”. Aunque es más apropiado decir que los Recursos son identificados mediante URIs. (Rafael Navarro, 2006)

Si solo existe una única URI como punto de acceso, posiblemente se esté creando un protocolo RPC. En tal caso, se debe desmenuzar el problema en tipos de recursos que se quieren manipular. Dos lugares donde se debe considerar cuando se buscan los recursos potenciales son las colecciones y las interfaces de búsqueda. Una colección de recursos es en sí mismo un recurso. Una interfaz de búsqueda también lo es, ya que el resultado de una búsqueda es otro conjunto de recursos. (Rafael Navarro, 2006)

A modo de ejemplo, supongamos un sistema de mantenimiento de una lista de contactos de empleados. En tal sistema cada usuario debería tener su propia URI con una apropiada representación. Además, la colección de recursos es otro recurso. (Rafael Navarro, 2006)

Por tanto, hemos identificado dos tipos de recursos, por tanto, habrá dos tipos de URIs:

- Employee (Una URI por empleado).
- AllEmployee (Listado de los empleados).

- ¿Cuál es el formato? Cuando hablamos informalmente de formato, estamos hablando de la representación. Como ya hemos comentado con anterioridad, no se puede acceder directamente al recurso, hay que obtener una representación. Esta representación puede ser un documento HTML, XML, una imagen, dependiendo de la situación. (Rafael Navarro, 2006)

Para cada uno de los recursos, se tiene que decidir cuál va a ser su representación. Si es posible, reutilizar formatos existentes, ayudará a incrementar la oportunidad de que nuestro sistema se componga con otros. (Rafael Navarro, 2006)

Continuando con nuestro ejemplo, el formato de representación va a ser XML, ya que es el principal formato para intercambio de información. El formato del empleado podría ser el siguiente, como se muestra en la figura 1:

```
<employee xmlns='HTTP://example.org/my-example-ns/'>
  <name>Full name goes here.</name>
  <title>Persons title goes here.</title>
  <phone-number>Phone number goes here.</phone-number>
</employee>
```

*Figura 1. Ejemplo básico estructura XML.*

*Fuente: <http://users.dsic.upv.es/~rnavarro/NewWeb/docs/RestVsWebServices.pdf>*

Para el listado de empleados podríamos tomar este otro de la figura 2:

```
<employee-list xmlns='HTTP://example.org/my-example-ns/'>
  <employee-ref href="URI of the first employee"/>
    Full name of the first employee goes here.</employee>

    <employee-ref href="URI of employee #2"/>Full
name</employee>
  .
  .
  <employee-ref href="URI of employee #N"/>Full
name</employee>
</employee-list>
```

*Figura 2. Representación básica de un archivo XML.*

*Fuente: <http://users.dsic.upv.es/~rnavarro/NewWeb/docs/RestVsWebServices.pdf>*

- ¿Qué métodos son soportados en cada URI? En este apartado se va definir cómo van a ser referenciados los recursos. Por medio de las URIs y los métodos soportados para hacer posible el acceso. El acceso se puede hacer de muchas formas, recibiendo una representación del recurso (GET o HEAD), añadiendo o modificando una representación (POST o PUT) y eliminando algunas o todas las representaciones (DELETE) como se puede especificar en la tabla 4. (Rafael Navarro, 2006)

HTTP	CRUD	Descripción
POST	CREATE	Crear un nuevo recurso
GET	RETRIEVE	Obtener la representación de un recurso
PUT	UPDATE	Actualizar un recurso
DELETE	DELETE	Elimina un recurso.

*Tabla 4. Comandos básicos de REST.*

*Fuente:*

*<http://users.dsic.upv.es/~rnavarro/NewWeb/docs/RestVsWebServices.pdf>*

Continuando con el ejemplo, ahora vamos a combinar recursos, representación y métodos, de acuerdo a la tabla 5:

Recurso	Método	Representación
Employee	GET	Formato del empleado
Employee	PUT	Formato del empleado
Employee	DELETE	-
All Employees	GET	Formato de la lista de empleados
All Employees	POST	Formato de empleado

*Tabla 5. Ejemplo de uso de REST.*

*Fuente:*

*<http://users.dsic.upv.es/~rnavarro/NewWeb/docs/RestVsWebServices.pdf>*

- ¿Qué códigos de estado pueden ser devueltos? No solo es necesario conocer qué tipo de representación va a ser devuelta, también es necesario enumerar los códigos de estado HTTP típicos que podrían ser devueltos, algunos de estos estados se pueden observar en la tabla 6. (Rafael Navarro, 2006)

Recurso	Método	Representación	Códigos de estado
Employee	GET	Formato del empleado	200, 301, 410
Employee	PUT	Formato del empleado	200, 301, 400, 410
Employee	DELETE	-	200, 204
All Employees	GET	Formato de la lista de empleados	200, 301

*Tabla 6. Códigos de estados de peticiones HTTP.*

*Fuente:*

*<http://users.dsic.upv.es/~rnavarro/NewWeb/docs/RestVsWebServices.pdf>*

## **DEBATE ENTRE LOS SERVICIOS WEB BASADOS EN REST Y SOAP**

Muchos diseñadores de Servicios Web están llegando a la conclusión que SOAP es demasiado complicado. Por tanto, están comenzando a utilizar Servicios Web basados en REST para mostrar cantidades de datos masivos. Este es el caso de grandes empresas como eBay y Google. (Rafael Navarro, 2006)

El problema principal surge del propósito inicial de SOAP. Esta tecnología fue originalmente pensada para ser una versión, sobre Internet, de DCOM o CORBA. Así lo demuestra su predecesor, el protocolo XML-RPC. Estas tecnologías lograron un éxito limitado antes de ser adaptadas. Esto es debido a que este tipo de tecnologías, las basadas en modelos RPC (Remote Procedure Call) son más adecuadas para entornos aislados, es decir, entornos donde se conoce perfectamente el entorno. La evolución en este tipo de sistemas es sencilla, se modifica cada usuario para que cumpla con los nuevos requisitos. (Rafael Navarro, 2006)

Pero cuando el número de usuarios es muy grande es necesario emplear una estrategia diferente. Se necesita organizar frameworks que permitan evolucionar, tanto por el lado del cliente como del servidor. Se necesita proponer un mecanismo explícito para la interoperabilidad de los sistemas que no poseen la misma API. Protocolos basados en RPC no son adecuados para este tipo de sistemas, ya que cambiar su interfaz resulta complicado. (Rafael Navarro, 2006)

Por esta razón, se intenta tomar como modelo a la Web. A primera vista se puede pensar que SOAP lo hace, ya que utiliza HTTP como medio de transporte. Pero Fielding argumenta que la Web funciona mejor cuando se utiliza en el estilo que lo hace REST. Utilizar HTTP como medio de transporte para protocolos de aplicación a través de firewalls es una idea equivocada. Esto reduce la efectividad de tener un firewall. Lo cual aumenta las posibilidades de nuevos agujeros de seguridad. (Rafael Navarro, 2006)

Sin embargo, los partidarios de SOAP argumentan que gracias a la tecnología existente que permite a los diseñadores encapsular la complejidad del sistema, dando lugar a interfaces generadas automáticamente que permiten facilitar el diseño del sistema. (Rafael Navarro, 2006)

Según argumenta Spolsky lo interesante de SOAP es que permite utilizar lenguajes de alto nivel para llamar y para implementar el servicio. Añade que “Alegar que SOAP es malo porque el formato de conexión es horrible y pesado es como alegar que nadie debería utilizar Pentiums porque su juego de instrucciones es mucho más complicado que el juego de instrucciones del 8086. Eso es cierto, pero por esa razón existen los compiladores.” (Rafael Navarro, 2006)

Las pautas a seguir en el diseño de servicios Web basados en REST son las mismas que las descritas en el apartado dedicado a crear una interfaz REST. Por otra parte, hemos ampliado dicha información en el contexto de los Servicios Web:

- Identificar todas las entidades conceptuales que se desean exponer como servicio.
- Crear una URL para cada recurso. Los recursos deberían ser nombres no verbos (acciones). Por ejemplo, no utilizar esto:

**`http://www.service.com/entities/getEntity?id=001`**

- Como podemos observar, `getEntity` es un verbo. Mejor utilizar el estilo REST, un nombre:

**`http://www.service.com/entities/001`**

- Categorizar los recursos de acuerdo con si los clientes pueden obtener una representación del recurso o si pueden modificarlo. Para el primero, debemos hacer los recursos accesibles utilizando un HTTP GET. Para el último, debemos hacer los recursos accesibles mediante HTTP POST, PUT y/o DELETE.
- Todos los recursos accesibles mediante GET no deberían tener efectos secundarios. Es decir, los recursos deberían devolver la representación del recurso. Por tanto, invocar al recurso no debería ser el resultado de modificarlo.
- Ninguna representación debería estar aislada. Es decir, es recomendable poner hipervínculos dentro de la representación de un recurso para permitir a los clientes obtener más información.

- Especificar el formato de los datos de respuesta mediante un esquema (DTD, W3C Schema). Para los servicios que requieran un POST o un PUT es aconsejable también proporcionar un esquema para especificar el formato de la respuesta.
- Describir como nuestro servicio ha de ser invocado, mediante un documento WSDL/WADL o simplemente HTML.

## CARACTERÍSTICAS DE REST Y SOAP, EN DEFINITIVA

El principal beneficio de SOAP recae en ser fuertemente acoplado, lo que permite poder ser analizado y depurado antes de poner en marcha la aplicación. En cambio, las ventajas de la aproximación basada en REST recaen en la potencial escalabilidad de este tipo de sistemas, así como el acceso con escaso consumo de recursos a sus operaciones debido al limitado número de operaciones y el esquema de direccionamiento unificado. (Rafael Navarro, 2006)

A modo de resumen, se puede apreciar las características de ambas aproximaciones en la siguiente tabla 7:

	SOAP	REST
<b>Características</b>	<ul style="list-style-type: none"> <li>• Las operaciones se definen en los mensajes.</li> <li>• Una dirección única para cada instancia del proceso.</li> <li>• Cada objeto soporta las operaciones estándares definidas.</li> <li>• Componentes débilmente acoplados.</li> </ul>	<ul style="list-style-type: none"> <li>• Las operaciones son definidas como puertos WSDL.</li> <li>• Dirección única para todas las operaciones.</li> <li>• Múltiples instancias del proceso comparten la misma operación.</li> <li>• Componentes fuertemente acoplados.</li> </ul>
<b>Ventajas declaradas</b>	<ul style="list-style-type: none"> <li>• Bajo consumo de recursos.</li> <li>• Las instancias del proceso son creadas explícitamente.</li> <li>• El cliente no necesita información de enrutamiento a partir de la URI inicial.</li> <li>• Los clientes pueden tener una interfaz "listener"</li> </ul>	<ul style="list-style-type: none"> <li>• Fácil (generalmente) de utilizar.</li> <li>• La depuración es posible.</li> <li>• Las operaciones complejas pueden ser escondidas detrás de una fachada.</li> <li>• Envolver APIs existentes es sencillo</li> <li>• Incrementa la privacidad.</li> <li>• Herramientas de desarrollo.</li> </ul>

	(escuchadora) genérica para las notificaciones. <ul style="list-style-type: none"> <li>• Generalmente fácil de construir y adoptar.</li> </ul>	
<b>Posibles desventajas</b>	<ul style="list-style-type: none"> <li>• Gran número de objetos.</li> <li>• Manejar el espacio de nombres (URIs) puede ser engorroso.</li> <li>• La descripción sintáctica/semántica muy informal (orientada al usuario).</li> <li>• Pocas herramientas de desarrollo.</li> </ul>	<ul style="list-style-type: none"> <li>• Los clientes necesitan saber las operaciones y su semántica antes del uso.</li> <li>• Los clientes necesitan puertos dedicados para diferentes tipos de notificaciones.</li> <li>• Las instancias del proceso</li> </ul>

Tabla 7. Comparación de REST vs SOAP.

Fuente: <http://users.dsic.upv.es/~mnavarro/NewWeb/docs/RestVsWebServices.pdf>

## DIFERENCIAS

La principal diferencia entre REST y SOAP se resume en los siguientes puntos de vista del propósito de la Web:

- **REST:** “La Web es el universo de la información accesible globalmente” (Tim Berners Lee)
- **SOAP:** “La Web es el transporte universal de mensajes”

A continuación, se van a plantear las diferencias entre REST y SOAP desde varios puntos de vista, como se muestra tabla 8:

	REST	SOAP
<b>Tecnología</b>	Interacción dirigida por el usuario por medio de formularios.	Flujo de eventos orquestados.
	Pocas operaciones con muchos recursos	Muchas operaciones con pocos recursos.
	Mecanismo consistente de nombrado de recursos (URI).	Falta de un mecanismo de nombrado.
	Se centra en la escalabilidad y rendimiento a gran escala para sistemas distribuidos hipermedia.	Se centra en el diseño
<b>Protocolo</b>	<pre> graph TD     POX[POX (Plain Old XML)]     subgraph HTTP     GET[HTTP GET]     POST[HTTP POST]     PUT[HTTP PUT]     DEL[HTTP DEL]     end     URI[Resource URI]     App[Application]     POX --- HTTP     HTTP --- URI     URI --- App           </pre>	<pre> graph TD     SOAP[SOAP (WS-*)]     subgraph Methods     SMTP[SMTP]     POST[HTTP POST]     MQ[MQ...]     end     URI[Endpoint URI]     App[Application]     SOAP --- Methods     Methods --- URI     URI --- App           </pre>

	XML autod descriptivo.	Tipado fuerte, XML Schema.
	HTTP.	Independiente del transporte.
	HTTP es un protocolo de aplicación.	HTTP es un protocolo de transporte
	Síncrono.	Síncrono y Asíncrono.
<b>Descripción del servicio</b>	Confía en documentos orientados al usuario que define las direcciones de petición y las respuestas.	WSDL.
	Interactuar con el servicio supone horas de testeo y depuración de URLs.	Se pueden construir automáticamente stubs (clientes) por medio del WSDL.
	No es necesario el tipado fuerte, si ambos lados están de acuerdo con el contenido.	Tipado fuerte.
	WADL propuesto en noviembre de 2006.	WSDL 2.0.
<b>Gestión del estado</b>	El servidor no tiene estado (stateless).	El servidor puede mantener el estado de la conversación.
	Los recursos contienen datos y enlaces representando transiciones a estados válidos.	Los mensajes solo contienen datos.
	Los clientes mantienen el estado siguiendo los enlaces.	Los clientes mantienen el estado suponiendo el estado del servicio.
	Técnicas para añadir sesiones: Cookies	Técnicas para añadir sesiones: Cabecera de sesión (no estándar)
<b>Seguridad</b>	HTTPS.	WS-Security.
	Implementado desde hace muchos años.	Las implementaciones están comenzando a aparecer ahora.
	Comunicación punto a punto segura.	Comunicación origen a destino segura.
	HTTPS.	WS-Security.

Tabla 8. Diferencias entre REST y SOAP.

Fuente: <http://users.dsic.upv.es/~mnavarro/NewWeb/docs/RestVsWebServices.pdf>

## 2.4 Json

JSON (JavaScript Object Notation) es un formato liviano de intercambio de datos. Es fácil para los humanos leer y escribir. Es fácil para las máquinas analizar y generar. Se basa en un subconjunto del lenguaje de programación JavaScript, estándar ECMA-262 3ª edición, diciembre de 1999. JSON es un formato de texto que es completamente independiente del lenguaje, pero utiliza convenciones que son familiares para los programadores de la familia C de idiomas, incluidos C, C ++, C #, Java, JavaScript, Perl, Python y muchos otros. Estas propiedades hacen de JSON un lenguaje ideal de intercambio de datos. (JSON, 2018)

JSON se basa en dos estructuras:

- ❖ Una colección de pares de nombre / valor. En varios idiomas, esto se realiza como un objeto, registro, estructura, diccionario, tabla hash, lista de claves o matriz asociativa.

- ❖ Una lista ordenada de valores. En la mayoría de los lenguajes, esto se realiza como una matriz, vector, lista o secuencia.

Estas son estructuras de datos universales. Prácticamente todos los lenguajes de programación modernos los respaldan de una forma u otra. Tiene sentido que un formato de datos que sea intercambiable con los lenguajes de programación también se base en estas estructuras. (JSON, 2018)

## 2.5 XML

**Extensible Markup Language (XML)** es un formato de texto simple y muy flexible derivado de SGML (ISO 8879). Originalmente diseñado para enfrentar los desafíos de la publicación electrónica a gran escala, XML también está desempeñando un papel cada vez más importante en el intercambio de una amplia variedad de datos en la Web y en otros lugares. (W3.org, 2018)

XML es un lenguaje de marcado creado por el World Wide Web Consortium (W3C) para definir una sintaxis para codificar documentos que los humanos y las máquinas podrían leer. Hace esto a través del uso de etiquetas que definen la estructura del documento, así como también cómo debe almacenarse y transportarse el documento. (Brady Gavin, 2018)

## 2.6 Microservicio

El estilo de arquitectura de microservicios es un enfoque para desarrollar una sola aplicación como un conjunto de pequeños servicios, cada uno ejecutándose en su propio proceso y comunicándose con mecanismos livianos, a menudo una API de recursos HTTP. Estos servicios se basan en las capacidades empresariales y se pueden implementar de forma independiente mediante una maquinaria de implementación completamente automatizada. Existe un mínimo de administración centralizada de estos servicios, que puede escribirse en diferentes lenguajes de programación y utilizar diferentes tecnologías de almacenamiento de datos. (James Lewis y Martin Fowler, 2014)

El estilo de microservicio, es útil compararlo con el estilo monolítico: una aplicación monolítica construida como una sola unidad. Las aplicaciones empresariales a menudo se construyen en tres partes principales: una interfaz de usuario del lado del cliente (que consiste en páginas HTML y JavaScript ejecutándose en un navegador en la máquina del usuario) una base de datos (que consiste en muchas tablas insertadas en una administración de base de datos común, generalmente relacional sistema) y una aplicación del lado del servidor. La aplicación del lado del servidor manejará las solicitudes HTTP,

ejecutará la lógica del dominio, recuperará y actualizará los datos de la base de datos, y seleccionará y completará las vistas HTML que se enviarán al navegador. Esta aplicación del lado del servidor es un *monolito*: un único ejecutable lógico. Cualquier cambio en el sistema implica construir y desplegar una nueva versión de la aplicación del lado del servidor. (James Lewis y Martin Fowler, 2014)

El estilo arquitectónico de microservicios: crear aplicaciones como suites de servicios. Además del hecho de que los servicios son de implementación independiente y escalables, cada servicio también proporciona un límite de módulo firme, incluso permitiendo que diferentes servicios se escriban en diferentes lenguajes de programación. También pueden ser administrados por diferentes equipos. (James Lewis y Martin Fowler, 2014)

- **MICROSERVICIOS Y VENTAJAS**

- Al ser cada microservicio independiente tenemos varias ventajas.
- Primero es más fácil para los desarrolladores hacerse con ellos ya que son relativamente “pequeños” y fáciles de abordar. (Cecilio Álvarez, 2015)
- Segundo si un microservicio falla no afecta a los demás ya que son totalmente independientes. (Cecilio Álvarez, 2015)
- Por otro lado, facilita el despliegue ya que no tenemos que desplegar aplicaciones gigantescas sino “microservicios” de tal forma que si solo tenemos que actualizar uno de ellos pues solo redespiegamos este. (Cecilio Álvarez, 2015)
- Por otro lado, y una de las cosas más importantes podemos acceder a estos microservicios desde todo tipo de dispositivo ya que los tenemos publicados vía REST o algún tipo de RPC. (Cecilio Álvarez, 2015)

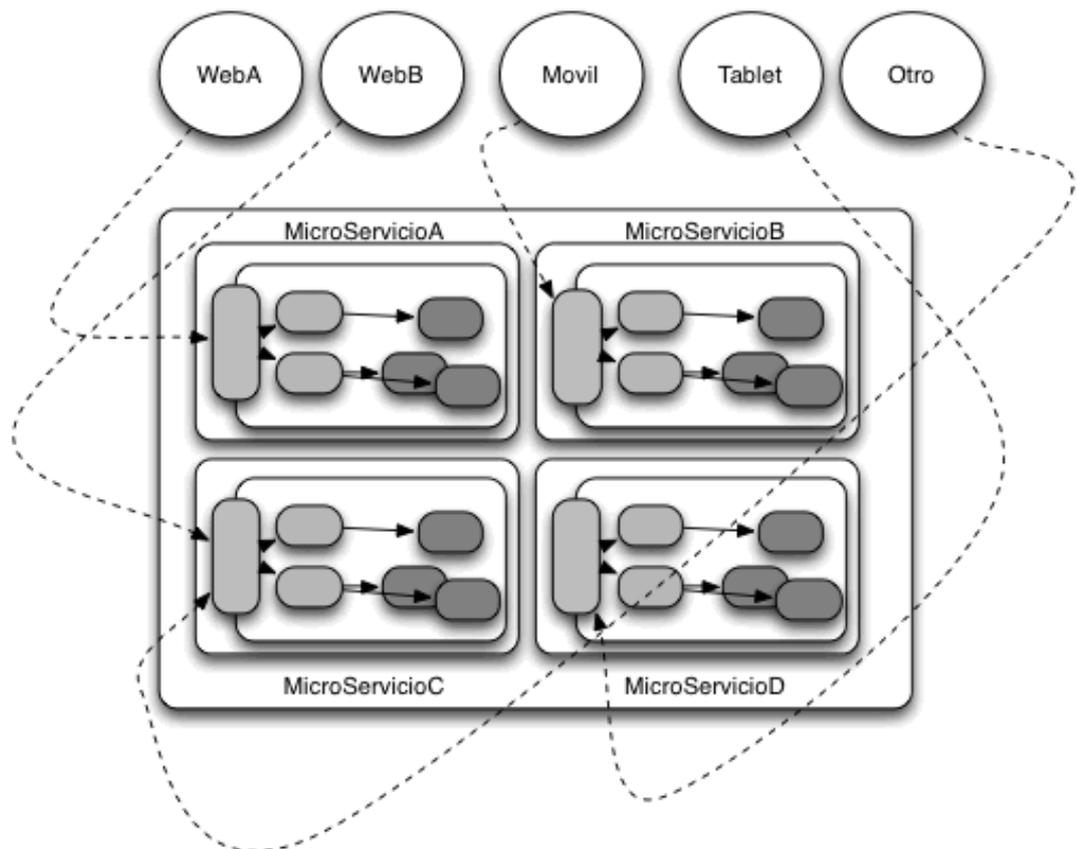


Figura 3. Esquema de uso de los microservicios.  
Fuente: <https://www.arquitecturajava.com>

## 2.7 SPA

Una aplicación de una sola página (SPA) es una aplicación web que se presenta al usuario a través de una única página HTML para que sea más receptiva y para replicar más de cerca una aplicación de escritorio o una aplicación nativa. Un SPA a veces se denomina interfaz de una sola página (SPI). (Juanda, 2012)

Una aplicación de una sola página puede recuperar todos los códigos HTML, JavaScript y CSS de la aplicación en la carga inicial o puede cargar recursos dinámicamente para actualizar en respuesta a la interacción del usuario u otros eventos. Otras aplicaciones web, en cambio, presentan al usuario una página inicial que está vinculada a partes de la aplicación en páginas HTML separadas, lo que significa que el usuario tiene que esperar a que se cargue una nueva página cada vez que realiza una nueva solicitud. (Juanda, 2012)

## Características de un SPA

- Por SPA se conocen las aplicaciones de una sola página o Single Page Applications.
- La aplicación se envía al navegador y la página no se recarga durante su uso.
- Una arquitectura SPA permite realizar cualquier aplicación tradicional de escritorio vía web, ya que el tiempo de respuesta es mucho más rápido que el de una aplicación web tradicional.

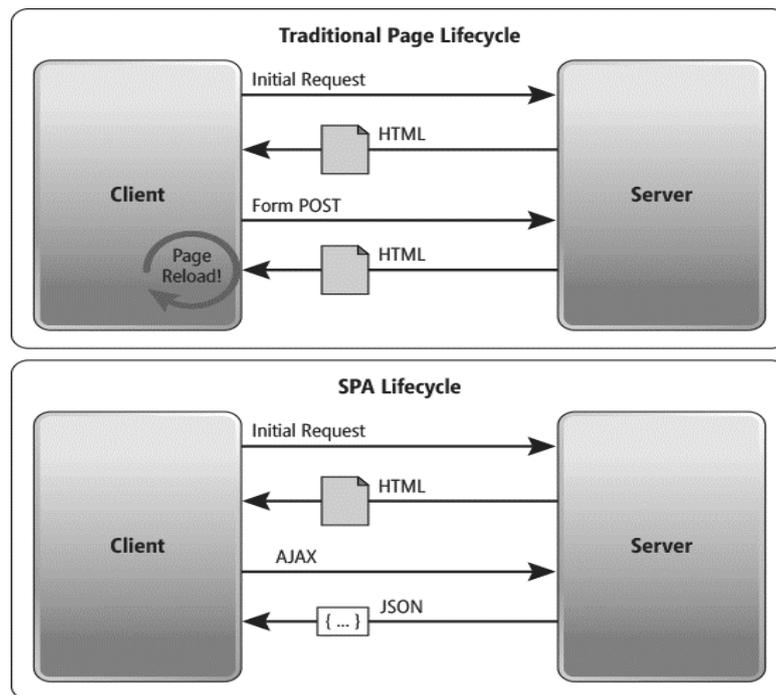


Figura 4. Características de SPA vs Web Tradicional.  
Fuente: <https://juanda.gitbooks.io/webapps/content/>

## Arquitectura de un SPA

- La mayor parte de la funcionalidad se lleva al cliente. Lo podríamos ver como un fat-client que se carga desde un servidor web.
- El código en servidor se usa básicamente para proveer de una API RESTful a nuestro código cliente usando Ajax.

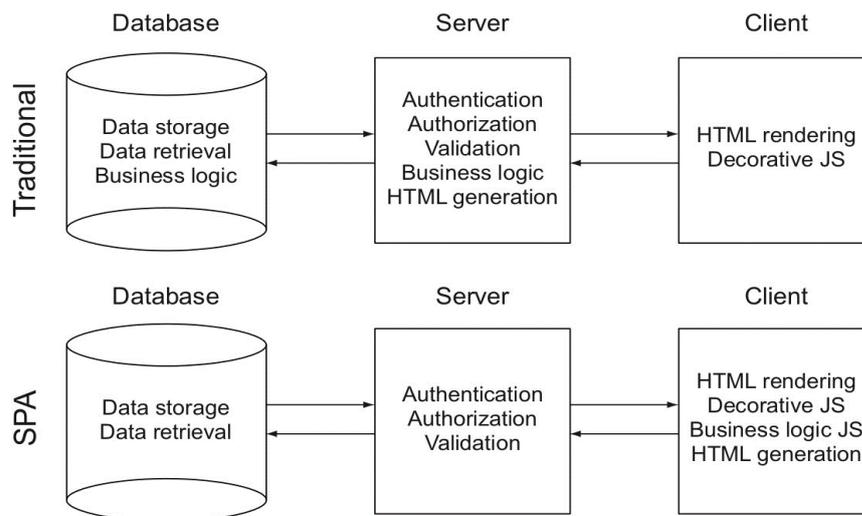


Figura 5. Arquitectura de SPA vs la Web Tradicional.  
Fuente: <https://juanda.gitbooks.io/webapps/content/>

### SPA vs App tradicional

- Si comparamos un SPA con desarrollos de aplicaciones de escritorio tradicionales, podemos ver como también hay numerosos puntos a favor de JavaScript:
  - El navegador es la aplicación más utilizada del mundo, ejecutar un SPA es tan sencillo como hacer un clic en una entrada en la barra de favoritos. El despliegue de la aplicación también es trivial
  - Es una solución multiplataforma
  - JavaScript se ha vuelto extremadamente rápido gracias a la competencia entre los distintos navegadores
- La propia evolución de JavaScript con el uso de JSON, jQuery, AJAX y muchas librerías actuales y potentes.
- Podemos utilizar JavaScript tanto en cliente como en servidor mediante Node.js.
- Existen bases de datos que pueden comunicarse directamente en JSON como CouchDB o MongoDB
- Dadas las ventajas inherentes a JavaScript hay herramientas para trabajar en otros lenguajes de programación y que generan posteriormente JavaScript. Un ejemplo es Google Web Toolkit (GWT) que genera JavaScript desde Java o Cappuccino que utiliza Objective-C. (Juanda, 2012)

## **2.8 Web dinámica**

Los sitios web dinámicos pueden cambiar el contenido de la página web de forma dinámica mientras la página se ejecuta en el navegador del cliente. Este tipo de sitios web usa programación del lado del servidor como PHP, Asp.NET, y JSP etc. para modificar el contenido de la página en tiempo de ejecución. Los sitios web dinámicos usan secuencias de comandos del lado del cliente para preparar el diseño dinámico y el código del lado del servidor para manejar el evento, administrar sesiones y cookies, y almacenar y recuperar datos de la base de datos. (Mangesh Bulkar, 2016)

## **2.9 Protocolo HTTP**

HTTP (protocolo de transferencia de hipertexto): es el conjunto de reglas para transferir archivos (texto, imágenes gráficas, sonido, video y otros archivos multimedia) en la World Wide Web. Tan pronto como un usuario de Web abre su navegador web, el usuario utiliza indirectamente HTTP. HTTP es un protocolo de aplicación que se ejecuta sobre el conjunto de protocolos TCP / IP (los protocolos básicos para Internet). (Margaret Rouse)

## 3 Estrategias de acceso a plataformas web dinámicas

En esta sección se mencionará toda la información relativa a las estrategias propuestas por los diferentes autores, los cuales plantean una solución para integrar plataformas web dinámicas, en base en esto se llevó a cabo la exploración de las diferentes tecnologías web más utilizadas hoy en día, para su posterior clasificación, análisis y selección de las herramientas de mejor popularidad para el desarrollo de software tanto del frontend como del backend.

### 3.1 Estado del arte

#### **Modelo de procesos para el desarrollo del frontend de aplicaciones web.**

**Autor:** José Jesús Valdivia Caballero

**Resumen:** La frase ‘divide y vencerás’, propone una estrategia de dividir una estructura de poder centralizada en grupos de poder más pequeños. Con base en este principio se propone el proyecto para desarrollar una aplicación web, donde pueda dividirse el proceso de implementación del aplicativo en procesos con sus herramientas, recursos propios y ciclo de vida. Tanto O’Reilly (2007) y Ginige (1998) hacen mención que las tecnologías a usar en front-end de una aplicación web son distintas a las del back-end, abriendo así la posibilidad de desacoplar la implantación en componentes propios del lado del cliente o browser web y del servidor de aplicaciones, bases de datos y archivos estáticos.

**Análisis:** la presente investigación hace un estudio referente al front-end y back-end donde su análisis busca que herramientas y tecnologías son más apropiadas para el desarrollo web en cuanto al uso del software libre, a su vez hace un enfoque a la utilización de frameworks para poder minimizar los procesos de desarrollo del lado del cliente con la ayuda de reutilización de componentes y hacer un uso correcto de sus funcionalidades (front-end), en cuanto al modelado de software hacen una orientación a la construcción de los modelos de la información, patrones, requerimientos y objetos para la elaboración de software con el uso de los microservicios que presenten alta disponibilidad, funcionalidad, compatibilidad, rendimiento, estabilidad, entre otros, que tendrá la logia del negocio y persistencia de los datos, referente al back-end se hace una comparativa de los lenguajes más apropiados para desarrollar aplicaciones web mostrando un ranking de las más utilizadas entre ellas tenemos JavaScript, Java, PHP, Python, Objective-C, C#, entre otras, las cuales son apropiadas para la arquitectura de los microservicios en cuanto al grupo de procesos que van a estar independientes ente el back-en y el fornt-end para los procesos de análisis y diseño aplicaciones web ágiles.

## **Diseño y desarrollo del backend para la explotación de componentes web, empleando google app engine**

**Autor:** Ana Isabel Lopera Martínez

**Resumen:** Existe una necesidad de medir la calidad de dichos desarrollos, para discernir si el concepto de componente web supone un cambio revolucionario en el desarrollo de la web 2.0. Para ello, es necesario realizar una explotación de componentes web, considerada como la medición de calidad basada en métricas y definición de un modelo de interconexión de componentes. La plataforma PicBit surge como respuesta a estas cuestiones. Consiste en una plataforma social de construcción de perfiles basada en estos elementos. Desde la perspectiva del usuario final se trata de una herramienta para crear perfiles y comunidades sociales, mientras que, desde una perspectiva académica, la plataforma consiste en un entorno de pruebas o *sandbox* de componentes web. Para ello, será necesario implementar el extremo servidor de dicha plataforma, enfocado a la labor de explotación, por medio de la definición de una interfaz REST de operaciones y un sistema para la recolección de eventos de usuario en la plataforma.

**Análisis:** el presente estudio se lleva a cabo una investigación sobre backend, el cual se va centralizar en desarrollar servicios con la API REST para ser consumido por GOOGLE APP ENGINE hacia la creación de componentes web utilizando CLOUD COMPUTING como herramienta de desarrollo, donde se desarrollará bajo la capa de presentación de contenidos (frontend) y conexiones a un servidor (backend) implementando la API, la cual utiliza un modelo de petición y respuesta HTTP que sigue App Engine. El desarrollo del backend lo hacen con ayuda de varias herramientas como Python o java, los cuales hacen uso de varias librerías para ofrecer los servicios cuando se hacen peticiones con métodos GET, POST, PUT, DELETE, entre otras, por medio de URL, esto se hace para llevar un enfoque flexible en una semántica REST que ofrece funcionalidades del servicio y operaciones cuando el cliente hace peticiones al servidor, además este documento presenta labores de testeo y depuración para el tiempo, status y cuerpo de las respuestas, mapeo entre recursos, funcionamiento de las alternativas y recolección de métricas en cuanto a la calidad de componentes web.

## **Proceso software de una aplicación web de apoyo a la oración.**

**Autor:** Thomas S. Birch

**Resumen:** Cada día, cristianos en todo el mundo hablan con Dios, de manera individual o junto con otros creyentes. La forma de orar no ha cambiado mucho

a lo largo del tiempo, y no depende de la tecnología. La manera en que se comunican las personas, sin embargo, sí ha experimentado grandes cambios. Es objetivo de este proyecto el desarrollar una aplicación web diseñada específicamente para gestionar motivos de oración. Se pretende facilitar la oración individual, sirviendo de alternativa a una libreta de papel en la que apuntar una lista de motivos, pero también la oración en comunidad, permitiendo que las peticiones de oración sean compartidas a través de Internet con otros creyentes, ya sea en grupos privados o haciéndolos públicos.

Para ello, se seguirá el proceso de desarrollo OpenUP, una variante simplificada del Rational Unified Process (RUP), y se hará uso de tecnología web actual: el framework Angular para desarrollar una aplicación SPA (Single Page Application) en Typescript que se ejecutará en el navegador, y algunos de los servicios que ofrece Firebase como backend; más concretamente, su base de datos en tiempo real y la gestión de la autenticación. En esta memoria de PFC, se describirá cómo se ha seguido el proceso software, se incluirán los productos de trabajo obtenidos como resultado, y se justificará la elección de la tecnología y las herramientas empleadas.

**Análisis:** este documento presenta una serie de tecnologías y procesos que se llevaron a cabo en la elaboración de una aplicación web que se basa en un grupo de oración (red social), para su desarrollo se utilizó angular como framework del lado del cliente (frontend) para poder ser accedida tanto en el navegador como a través de una aplicación móvil, a su vez se hace uso de java en el back-end, con un framework como Spring MVC, el cual se utilizara con los servicios REST para ofrecerle al cliente las funcionalidades de la aplicación el cual va acceder a los datos que se encuentren en el motor de base de datos MySQL o NoSQL, esto se llevó a cabo en procesos por fases que se desarrolló bajo UML y poder hacer productos de trabajo ágiles y fáciles de utilizar.

### **Aplicación Android de recetas de cocina con backend symfony.**

**Autor:** José María Moreno Fernández-Cañadas

**Resumen:** El presente trabajo de fin de máster pretende construir una plataforma web de recetas de cocina que incorpore las buenas prácticas de la web 2.0. Con ese objetivo, el servidor se implementa como un servicio web que aplica una arquitectura *REST* sobre la cual se podrán realizar peticiones desde cualquier cliente independientemente de su tecnología, eliminando de esta manera las restricciones en cuanto a dispositivo. Además, para facilitar su uso como servicio, se incorpora al proyecto la especificación de la API con la intención de servir como fuente de documentación en el caso de que se quieran integrar aplicaciones o clientes de terceros. Por otra parte, y para completar la

aplicación, de entre todas las posibilidades existentes, se elige *Android* para la implementación en la capa cliente y para profundizar de esta manera en dicha tecnología. En cuanto a la aplicación, se basa por completo en las aportaciones de los usuarios, que aportan recetas, etiquetas y comentarios, y que enriquecen el contenido de la aplicación al mismo tiempo que lo consumen. Para el desarrollo de la aplicación se sigue una metodología web ligera basada en iteraciones breves donde se aplican distintas disciplinas de desarrollo de software para desarrollar cada una de las funcionalidades detectadas.

**Análisis:** este documento presenta un enfoque sobre backend, el cual se llevara a cabo para desarrollar una aplicación web 2.0, según el autor presenta en su investigación y desarrollo un objetivo general sobre los servicios web para el uso de la arquitectura REST, donde busca eliminar la limitaciones que presentan actualmente el software web (no más de una tecnología de desarrollo), siguiendo la documentación que estipula la API en este caso de REST servirá como integración de aplicaciones y clientes de terceros. Para llevar a cabo su desarrollo se basa en arquitectura de patrones de diseño, frontend para móviles, backend con tecnologías de libre distribución y metodologías de desarrollo web el cual buscara tener un desarrollo ágil y ligero para su implantación, donde utilizara un framework como lo es Symfony (desarrollado en PHP) del lado del servidor.

### **Arquitectura de software basada en microservicios para desarrollo de aplicaciones web.**

**Autor:** Daniel López, Edgar Maya

**Resumen:** Actualmente, el proceso de desarrollo de software que realiza la Coordinación General de Tecnologías de la Información y Comunicación (CGTIC) de la Asamblea Nacional del Ecuador (ANE) constituye el empleo de una arquitectura de software tradicional o monolítica que ha sido adoptada del lenguaje de programación utilizado, la plataforma o de la experiencia del personal del área de desarrollo; por el aspecto monolítico, este tipo de aplicaciones empaquetan toda la funcionalidad en una sola y gran unidad ejecutable (un solo archivo o aplicación), lo que ha provocado dificultades en aspectos como mantenimiento, escalabilidad y entregas. El objetivo del presente estudio fue identificar las tecnologías, metodología y arquitectura que utiliza la CGTIC para el desarrollo de aplicaciones web y la correspondiente identificación de las tecnologías existentes para el desarrollo e implementación de microservicios, utilizando como base de la investigación un enfoque cualitativo, con un tipo de investigación descriptiva y diseño documental. Se empleó la técnica de grupo focal aplicado a los funcionarios del área de desarrollo de

software de la CGTIC, revisión bibliográfica de arquitectura de microservicios. Como avance de la investigación, el análisis ha permitido identificar el estado del arte respecto a microservicios y su implementación, así como la identificación de los requisitos y necesidades relativos al desarrollo de aplicaciones web y como satisfacerlas mediante el diseño de una arquitectura de software.

**Análisis:** el presente artículo se basa en un investigación referente al uso y existencias de las tecnologías para el desarrollo e integración de tecnologías web, el cual hace un análisis de las diferentes herramientas que existe para el desarrollo web, tomado como base los microservicios, la cual se orienta en un conjunto de pequeños servicios para la comunicación de aplicaciones a través del protocolo HTTP, esto trae una serie de topologías para su utilización, algunas de ellas son API REST y SOAP estas se basan en servicios web, pero son tecnologías web totalmente diferentes, en el que se da una breve aclaración de sus diferencias y usos actualmente a nivel global.

### **Sistema de filtrado web inteligente basado en un sistema multi-agente.**

**Autor:** Emba Hernández José Manuel

**Resumen:** Desarrollar un Sistema de filtrado de contenidos Web capaz de evitar el uso excesivo y mal intencionado de la Web dentro de un hogar, una empresa o cualquier institución, dependiendo de las características de cada ambiente. Desarrollando el sistema aprovechando las potencialidades de las capacidades de interoperabilidad de los servicios Web y la automatización de procesos por medio de los agentes inteligentes. Añadiendo la capacidad de que algún proveedor o usuario pueda limitar los servicios en bases de datos o análisis

**Análisis:** esta tesis presenta un estudio sobre el filtrado web (navegación segura, productividad, contenido inapropiado, sobrecarga de información, etc..) para las empresas, escuelas y personas que trabajen en sitios web que no contienen ningún tipo de restricción, para ello el autor pretende hacer uso de los servicios web para crear filtros en las peticiones del lado del cliente con la ayuda de agentes para controlar el tiempo de la intervención humana cuando se hagan búsquedas que solucionen la interoperabilidad entre aplicaciones web, para esto el autor utilizo servicios web (web Services) con una arquitectura SOA que se basa en las diferentes capas de comunicación de datos utilizando lo diferentes protocolos que existen actualmente, también hace énfasis en la arquitectura REST que es otra alternativa para la construcción de servidos web la cual solo hace uso del protocolo HTTP, logrando evitar otros protocolos que utiliza SOAP, esto lo lleva hacer un análisis de factibilidad de los lenguajes de programación que son más usados por las arquitecturas anteriormente descritas, donde se

encuentra que las tecnologías más apropiadas son Java, PHP, C# y Python para la creación de servicios web.

**Clubmat 1.1: extensión de una aplicación web destinada al fortalecimiento de clubes escolares matemáticos integrando javafx2 y javaee6 con servicios web basados en REST.**

**Autor:** Julián Camilo Ortega Muñoz

**Resumen:** En el desarrollo de aplicaciones web empresariales existe gran variedad de tecnologías para su realización. Una tecnología importante es JavaEE6. En la arquitectura JavaEE6 se trabaja como estándar el framework JSF (*Java Server Faces*) para la realización de interfaces de usuario. Este framework presenta una serie de problemáticas en cuestiones de seguridad, interactividad, tiempo de desarrollo, entre otras. En la necesidad de resolver estas problemáticas se exploró JavaFX2, un framework RIA (*Rich Internet Applications*) que por sus características soluciona ciertos aspectos en la interacción del usuario con la aplicación empresarial y es menos susceptible de ser alterado en ejecución por usuarios mal intencionados. Previamente se realizó un trabajo de integración entre estas tecnologías (JavaFX2 y Java-EE6), utilizando servicios web basados en SOAP. Este trabajo fue basado en un caso de estudio enmarcado en el programa social PROSOFI de la Pontificia Universidad Javeriana acerca de los clubes de matemáticas de la comunidad de Usme y se desarrolló parte del sistema de información denominado CLUBMAT. CLUBMAT es un sistema de información web que gestiona y administra clubes de matemáticas y fue el caso de estudio para el trabajo de integración entre JavaFX2 y JavaEE6. El propósito de este trabajo de grado fue estudiar la arquitectura dejada previamente, en la realización de este software para encontrar mejoras en el acoplamiento de las tecnologías, explorando los servicios web basados en REST. Así mismo, dejar un estándar de la integración de estas tecnologías y complementar el sistema CLUBMAT en su totalidad respecto a los requerimientos del cliente. Lo anterior favoreció para la formulación de conclusiones, con las cuales queremos ayudar a nuestro público objetivo, a obtener una visión general de las condiciones actuales de la aplicación de CLUBMAT, de manera que se pueda dar una implementación rápida y útil de la misma, y así mismo la permanencia y réplica de la aplicación a otras comunidades con una necesidad similar.

**Análisis:** este trabajo se basa en la creación de una aplicación web para estudiantes de un club de matemáticas, el autor se basa en la utilización de java del lado del servidor (backend), para ello se fundamentó en las arquitecturas REST y SOAP para la creación de servicios web que puedan ser consumidos por los clientes, en este caso los estudiantes del club, a su vez se hace una comparación entre REST vs SOAP para ver qué arquitectura es más fácil de

trabajar, donde se obtuvo el resultado que REST es mucho más factible que SOAP ya que trabaja un solo protocolo de comunicación (HTTP) en cambio SOAP trabaja varios protocolos de comunicación de las capas OSI.

## **Web application front-end architecture and development using AngularJS framework.**

**Autor:** Stanislav Nazmutdinov

**Abstract:** The aim of this thesis is to investigate Single Page Web Application front-end development and architecture that is developed with AngularJS JavaScript framework. Both Single Page Application concept and AngularJS framework are relatively new phenomena in web application development, therefore their features need to be examined.

As the result of the work there will be developed a property rental web application. During the project development will be reviewed various AngularJS-based application features. Project development will not be limited only to AngularJS framework. AngularJS is pretty versatile and flexible and therefore provides many ways to use and integrate other JavaScript libraries as well as utilize various JavaScript development tools. Despite the fact that the main goal of this thesis is to examine application front-end development with AngularJS there will be developed both front end and server-side of the application and briefly described application back end.

**Resumen:** El objetivo de esta tesis es investigar el desarrollo y la arquitectura frontend de la aplicación web de una sola página que se desarrolla con el marco de AngularJS JavaScript. Tanto el concepto de aplicación de página única como el framework AngularJS son fenómenos relativamente nuevos en el desarrollo de aplicaciones web, por lo tanto, es necesario examinar sus características. Como resultado del trabajo, se desarrollará una aplicación web de alquiler de propiedades. Durante el desarrollo del proyecto se revisarán varias características de aplicaciones basadas en AngularJS. El desarrollo del proyecto no estará limitado solo al marco AngularJS. AngularJS es bastante versátil y flexible y, por lo tanto, proporciona muchas formas de usar e integrar otras bibliotecas de JavaScript, así como utilizar varias herramientas de desarrollo de JavaScript. A pesar de que el objetivo principal de esta tesis es examinar el desarrollo del frontend de la aplicación con AngularJS, se desarrollarán tanto la parte frontal como la del lado servidor de la aplicación y la parte posterior de la aplicación se describe brevemente.

**Análisis:** este trabajo de investigación hace énfasis en el frontend para el desarrollo de aplicaciones SPA de una sola página donde el autor hace uso del

framework AngularJS el cual se utilizará para la creación de un aplicación de un alquiler de propiedades, en el trascurso de trabajo hace un estudio de comparativo entre los framework más utilizados hasta el 2015 donde los resultados arrojan que el más utilizado es AngularJS, a su vez hace un modelo de integración de tecnologías web para consumir recursos o servicios web que se encurtan alojados en el servidor, para ello hace uso de las librerías que tiene el framework AngularJS para el Controlador y vista del lado del cliente de la aplicación web.

## COMPARATIVO

Titulo	Autor	Tecnologías usadas
<b>FRONTEND</b>		
MODELO DE PROCESOS PARA EL DESARROLLO DEL FRONT-END DE APLICACIONES WEB	José Jesús Valdivia Caballero	<p>En esta investigación el autor indica las herramientas que se deben utilizar en el proceso de elaboración del frontend para la etapa de construcción, estas son: <b>Java, PHP, Python, Ruby, Go</b>, a su vez da una aclaración de las herramientas que se deben utilizar en las diferentes etapas del grupo de proceso de elaboración del desarrollo web.</p> <p>La estrategia que propone este artículo consiste en divide y venceras, para el grupo de procesos, en el desarrollo frontend de bajo acoplamiento con el backend que tendrá la lógica de negocio, donde se centrará en la arquitectura de microservicios para separar el cliente del servidor y así mejorar los tiempos de respuesta. (Pág. 201-202)</p>
PROCESO SOFTWARE DE UNA APLICACIÓN WEB DE APOYO A LA ORACIÓN	Thomas S. Birch	<p>En la parte del frontend el autor hace énfasis en el uso de los <b>frameworks</b>, ya que estos permiten ahorrar tiempo a la hora de desarrollar y estructurar mejor el código, a su vez se invita hacer uso de los siguientes <b>frameworks Backbone.js, AngularJS, EmberJS, ReactJS, Vue.js</b> ya que manejan un patrón MVC, de los cuales <b>AngularJS</b> se toma como elección para desarrollar su proyecto, ya que posee las características de multiplataforma, velocidad y rendimiento, productividad, entre otras.</p>
WEB APPLICATION FRONT END ARCHITECTURE AND DEVELOPMENT USING ANGULARJS FRAMEWORK	Stanislav Nazmutdinov	<p>Este artículo presenta un desarrollo de una aplicación web orientada a servicios, donde el autor hace un análisis de los diferentes <b>frameworks</b> que son utilizados hoy en día, encontrando que <b>AngularJS</b> es el más popular para el desarrollo de frontend.</p>
<b>BACKEND</b>		
DISEÑO Y DESARROLLO DEL BACKEND PARA LA EXPLOTACIÓN DE COMPONENTES WEB, EMPLEANDO	Ana Isabel Lopera Martínez	<p>En este trabajo se hace uso de las tecnologías <b>REST, Java y Python</b> del lado del servidor, las cuales van hacer integradas con <b>GOOGLE APP ENGINE</b> para el desarrollo del lado del cliente.</p>

GOOGLE APP ENGINE		
PROCESO SOFTWARE DE UNA APLICACIÓN WEB DE APOYO A LA ORACIÓN	Thomas S. Birch	Del lado del backend el autor hace uso de la tecnología <b>REST</b> como parte inicial a su desarrollo, el cual va ser manipulado por el <b>framework Spring MVC</b> , que está desarrollado bajo <b>Java</b> .
APLICACIÓN ANDROID DE RECETAS DE COCINA CON BACKEND SYMFONY	José María Moreno Fernández-Cañadas	Los autores de esta tesis hacen énfasis en del desarrollo del lado del servidor, para ello hacen uso de la arquitectura <b>REST</b> como tecnología principal para el <b>desarrollo móvil</b> con <b>Android Studio</b> , para ser accedido a través del <b>framework SYMFONY</b> cuando se hagan peticiones al servidor.
ARQUITECTURA DE SOFTWARE BASADA EN MICROSERVICIOS PARA DESARROLLO DE APLICACIONES WEB	Daniel López, Edgar Maya	Este artículo hace un enfoque hacia la arquitectura de <b>microservicios</b> como una alternativa al desarrollo de aplicaciones web que utilizan las tecnologías <b>REST</b> y <b>SOAP</b> .
SISTEMA DE FILTRADO WEB INTELIGENTE BASADO EN UN SISTEMA MULTI-AGENTE	Emba Hernández José Manuel	Esta tesis hace un enfoque hacia las arquitecturas <b>REST</b> y <b>SOAP</b> para la creación de servicios web del lado del servidor, donde resalta las tecnologías más apropiadas para el desarrollo, entre ellas tenemos: <b>Java, PHP, C# y Python</b>
CLUBMAT 1.1: EXTENSIÓN DE UNA APLICACIÓN WEB DESTINADA AL FORTALECIMIENTO DE CLUBES ESCOLARES MATEMÁTICOS INTEGRANDO JAVAFX2 Y JAVAEE6 CON SERVICIOS WEB BASADOS EN REST	Julián Camilo Ortega Muñoz	En esta tesis el autor utilizo las tecnologías de <b>JAVAFX</b> y <b>JVAEE6</b> para el desarrollo de su trabajo orientado a servicios web basados en la arquitectura <b>REST</b> .

*Tabla 9. Análisis de las diferentes tecnologías backend y frontend  
Fuente: Autor del proyecto.*

**Apreciación:** se puede determinar que existen varias tecnologías web de desarrollo, tanto de frontend como del backend, asimismo se logró descubrir que el uso de frameworks trae grandes ventajas para un desarrollo ágil y rápido en la construcción de aplicaciones web, a su vez las tecnologías más utilizadas por los autores mencionados anteriormente del lado del cliente, encontramos que se utilizan frameworks como; Backbone.js, AngularJS, EmberJS, ReactJS, Vue.js, destacando como el más utilizado actualmente AngularJS para el desarrollo frontend, esto permite a este trabajo hacer uso de cualquier tecnología, para las

alternativas en desarrollo del lado del servidor, se encontró que los autores sugieren usar las tecnologías más apropiadas como; Java, PHP, C# y Python, en donde la mayoría utiliza java como herramienta principal de desarrollo web, asimismo se recomienda hacer uso de las arquitecturas SOAP y REST, las cuales se basan en los servicios web, por lo que se resalta actualmente REST, por su fácil uso según el análisis de los autores.

### **3.2 Modelos propuestos de la arquitectura de microservicios**

En esta parte del documento se describe un análisis comparativo de algunos modelos propuestos para la integración de tecnologías web, ya que estos son fundamentales para el planteamiento de la estrategia propuesta, para llevar a cabo el objetivo principal de este trabajo.

- **Modelo: patrón: api Gateway/ backend para frontend.**

Referente al artículo: patrón: API Gateway/ Backend para frontend, el autor Chris Richardson, nos brinda una breve introducción al contexto del funcionamiento de los microservicios en cuanto a la comunicación en la interfaz de usuario y el servidor, para ello el autor da un claro ejemplo de una aplicación en línea, la cual expone una serie de interacciones con las diferentes transacciones que hace la aplicación, pero existen grandes inconvenientes cuando la aplicación es migrada a varias plataformas ya sean Móviles, Navegadores Web, Tablet, entre otras, esto hace que el desarrollo sea bastante complejo y lento a la hora de su construcción.

Según la problemática anteriormente descrita, el autor hace énfasis en el uso de los servicios web, ya que la utilización del patrón de arquitectura de microservicios, divide las diferentes funcionalidades en múltiples servicios, esto trae un gran interrogante en su artículo, ya que se pregunta; ¿Cómo los clientes acceden a la aplicación basada en pequeños servicios ubicados de forma individual?

La solución más efectiva que propone el autor es implementar una puerta de enlace API, para las diversas peticiones que hace el cliente al servidor, esto diferenciará a la arquitectura actualmente utilizada donde todo se encuentra en una sola aplicación tanto el cliente como el servidor (arquitectura monolítica), para dar solución a lo anterior el autor propone dividir el cliente y el servidor en dos partes separadas e implementar una puerta de enlace que opera las solicitudes de los clientes de dos formas una a través de un proxy

o enrutadas al servidor, y dar acceso al frontend y al backend para comunicarse, como se muestra en la figura 6.

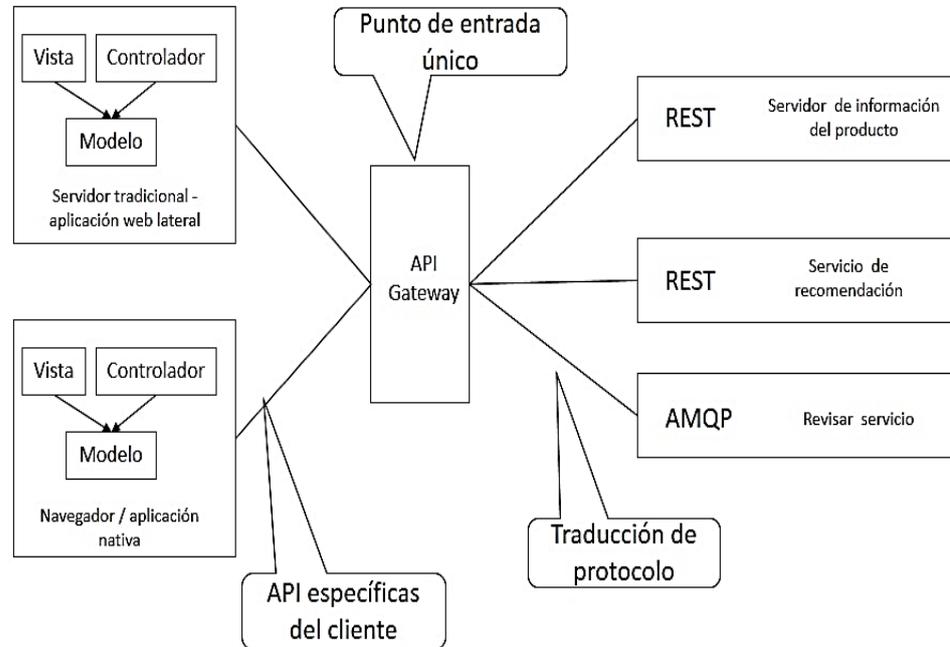


Figura 6. Puerta de enlace API para comunicar cliente-servidor.  
Fuente: <https://microservices.io>

**Nota:** según la figura 6, se puede apreciar la separación tanto del cliente como del servidor, donde el canal de acceso va ser la puerta de enlace API como protocolo de comunicación del lado del frontend que utiliza la arquitectura Modelo-Vista-Controlador con el backend que opera con diferentes topologías (REST, AMQP, ...) para operar los datos de las diferentes solicitudes que se hacen al servidor y viceversa.

Este modelo propone una comunicación intermedia entre en backend y el frontend, esto se hace con el fin de tener varias puertas de enlace para dar acceso a los servicios de varias formas, como se muestra en la figura 7.

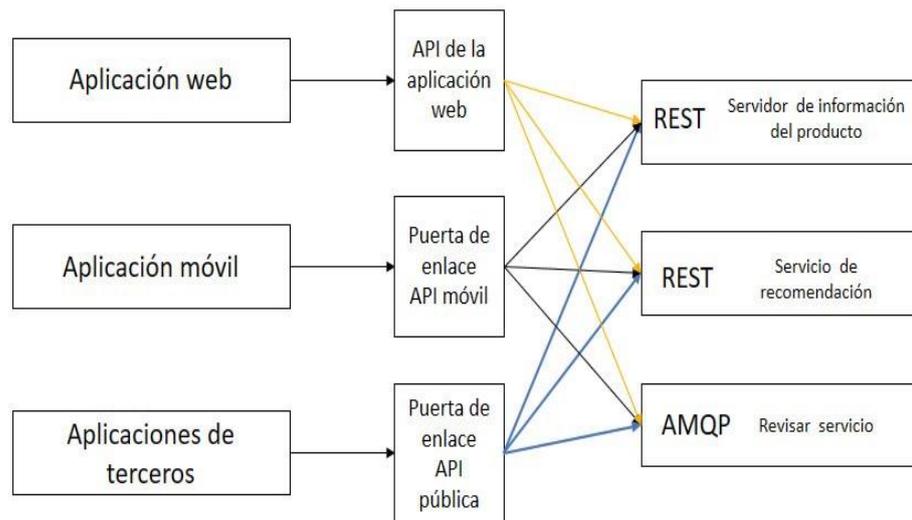


Figura 7. Variación: backend para frontend.  
Fuente: <https://microservices.io>

Se puede apreciar el patrón de la arquitectura de microservicios, define una puerta de enlace para las diferentes plataformas que van acceder a través del API Gateway a los servicios que serán ofrecidos por el backend. (Chris Richardson, 2017)

- **Modelo de integración de tecnologías web para la gestión de contenido virtual b2b.**

Otra de las estrategias que se va analizar hace referente al modelo que plantean los autores de del artículo Modelo de integración de tecnologías web para la gestión de contenido virtual B2B, el cual se plantea por el constante avance que tiene hoy en día internet, en cuanto al comercio electrónico, el cual requiere soluciones que cubran grandes necesidades automáticas a las empresas que brinda este tipo de servicios a los usuarios para los procesos de pagos, búsqueda de productos (según su preferencia), administración de ventas, entre otras, para esto actualmente existen varias tecnologías que brindan soporte a lo anteriormente descrito algunas de ellas son los servicios web, agentes, modelado virtual, entre otras, las cuales trabajan independientemente en aspectos como interoperabilidad, automatización de procesos, estandarización, apoyo en la toma de decisiones y formas de presentación de información, etc.

Para dar solución a lo anteriormente descrito por los autores de este artículo, hacen énfasis en una propuesta que permita la integración de las tecnologías descritas en su investigación, ya que cada una de ellas brinda grandes

ventajas y características fundamentales para la comunicación de los modelos de negocio.

La propuesta consiste en construir un modelo que permita agrupar las características de cada una de las tecnologías y facilite la eficiente administración adecuada de la información, en este caso contenido virtual aplicado al comercio electrónico.

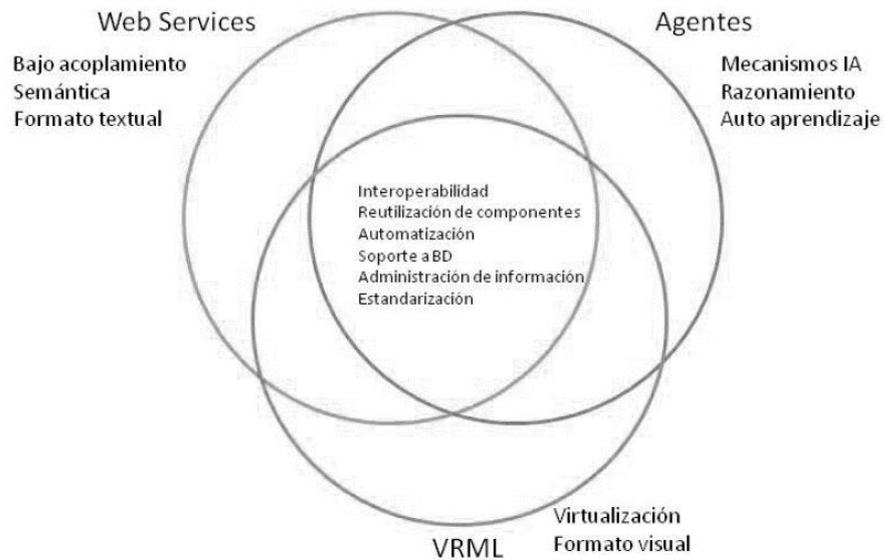


Figura 8. Modelo de integración de ventajas tecnológicas.  
Fuente: <https://www.researchgate.net/>

La figura 8 muestra una introducción al modelo que se va a adaptar a la integración, ya que toma las ventajas y características que tiene las diferentes tecnologías que se van a utilizar para agruparlas en una sola, este análisis permitió hacer una agrupación de las ventajas y características más apropiadas de cada tecnología para la construcción de la estrategia, con base a esto los autores diseñaron el siguiente modelo:

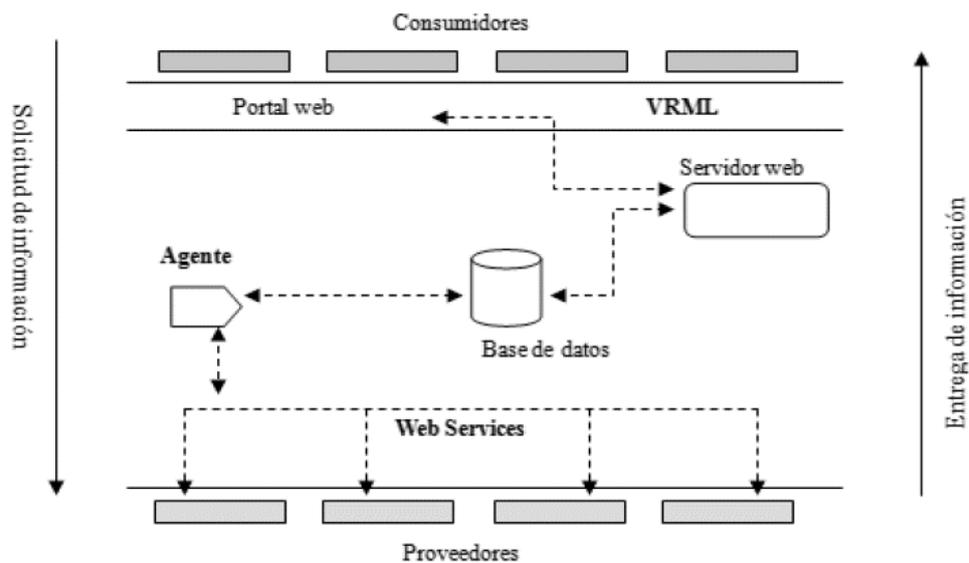


Figura 9. Prototipo de integración de tecnologías.  
Fuente: <https://www.researchgate.net/>

El prototipo de la figura 9, establece una integración de las tecnologías mencionadas anteriormente, este modelo describe el funcionamiento de la arquitectura de microservicios, ya que separa tanto el frontend como el backend. Logrando diferenciar que el frontend comprende toda la capa de los consumidores y el portal web y en cuanto al backend comprende la capa de los proveedores o servicios web que serán encargados de los datos según el filtro del agente el cual hace el rol de recibir y transmitir las funcionalidades de las operaciones cuando se hace una transacción del cliente al servidor, este modelo se puede adaptar a cualquier tipo de aplicación web ya que utiliza microservicios para la lógica de negocio. (José Raymundo - Abelardo Rodríguez - Héctor Andrade - Rafael Rivera, 2013)

- **Modelo: arquitectura basada en microservicios (estilo orquestado).**

La arquitectura basada en microservicios de la investigación de Manuel Pérez se basa en una comparación entre las arquitecturas monolíticas (aplicaciones web tradicionales) vs microservicios (servicios web pequeños distribuidos individualmente con un funcionamiento específico), el autor especifica que la arquitectura monolítica tiene muchas desventajas cuando se van hacer cambios en el servidor, como por ejemplo; si se hace un cambio en el código este puede afectar todo el código de la aplicación web y afectaría con errores los procesos de compilación del software, por lo que puede alterar costos, tiempos, soporte, etc. Para esto el autor considera que es necesario introducir

el desarrollo de la arquitectura de microservicios en el desarrollo de aplicaciones web, ya que esta trae grandes avances en cuanto a la monolítica, uno de estos avances consiste en la separación de los componentes de la aplicación web en pequeños servicios web distribuidos individualmente en diferentes servidores interconectados entre ellos.

Referente a lo anterior el autor lo define según los conceptos de los investigadores James Lewis y Martin Fowler de su artículo “**Microservices**”, donde ellos definen el término microservicio como un “estilo arquitectural en el que múltiples servicios, cada uno corriendo de manera individual y desplegados de la forma más automatizada posible, se comuniquen entre sí mediante mecanismos ligeros”, de aquí parte el autor para llevar a cabo un modelo que se acople al uso de los microservicios, para ello se va a basar en las características de dicha arquitectura, en cuanto a las ventajas y desventajas de su implementación.

La visión global que propone el autor se basa en dos estilos: uno de ellos es el estilo orquestado el cual consiste en tener un monitor de eventos que actúa como director de orquesta a los servicios web, el cual se encarga de dirigir y manejar las peticiones de los clientes a los servicios web que estén encargados de brindar las respuestas de los datos requeridos, el segundo es el estilo coreográfico que consiste en un único coordinador que va decidir si la petición se ejecutara o no, el cual informa a todos los microservicios y el encargado brinda la petición. De lo anterior descrito el autor sugiere que los dos estilos son apropiados para el desarrollo de aplicaciones web orientadas a servicios, en su comparación difiere que el segundo estilo es más complejo de utilizar ya que utiliza un evento complejo que consiste en recorrer todos los servicios al tiempo y decidir cuál es el que brindara los resultados, por lo que el otro estilo solo se dirige al servicio que va ser utilizado, el autor concluye que el estilo orquestado es más fácil de implementar.

De lo anteriormente descrito el autor se enfoca en un **estilo orquestado**, presentado el siguiente modelo que muestra un esquema del uso de la arquitectura logia del despliegue independiente de los microservicios, como se muestra a continuación:

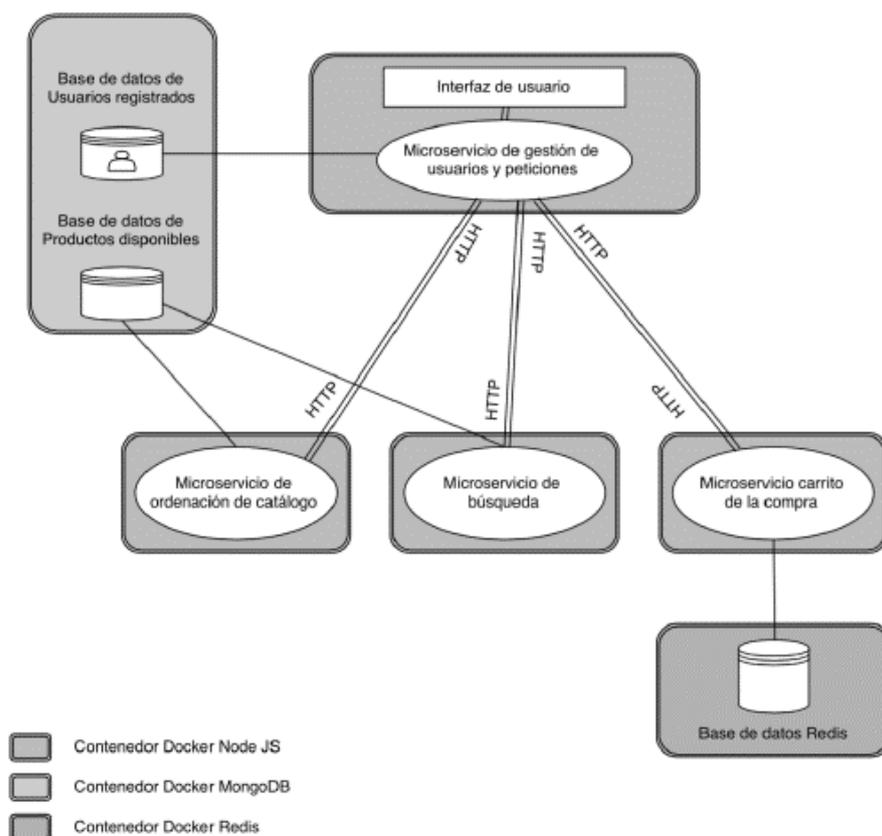


Figura 10. Esquema de una arquitectura lógica de microservicios.  
Fuente: <http://oa.upm.es/37346/>

Se puede apreciar en la figura 10, la separación del frontend y el backend, como se modela en la figura el frontend hace referente a la interfaz de usuario y el backend al despliegue de los servicios web, los cuales serán comunicados por el protocolo HTTP con las diferentes peticiones que se hagan por el cliente, también se puede ver el uso del estilo orquestado donde el director de eventos hace el rol de un microservicio de gestión de usuarios y peticiones a los otros servicios del lado de servidor para brindar las transacciones a los clientes.

La funcionalidad de los servicios web desplegados funciona de manera independiente, bajo el siguiente esquema:

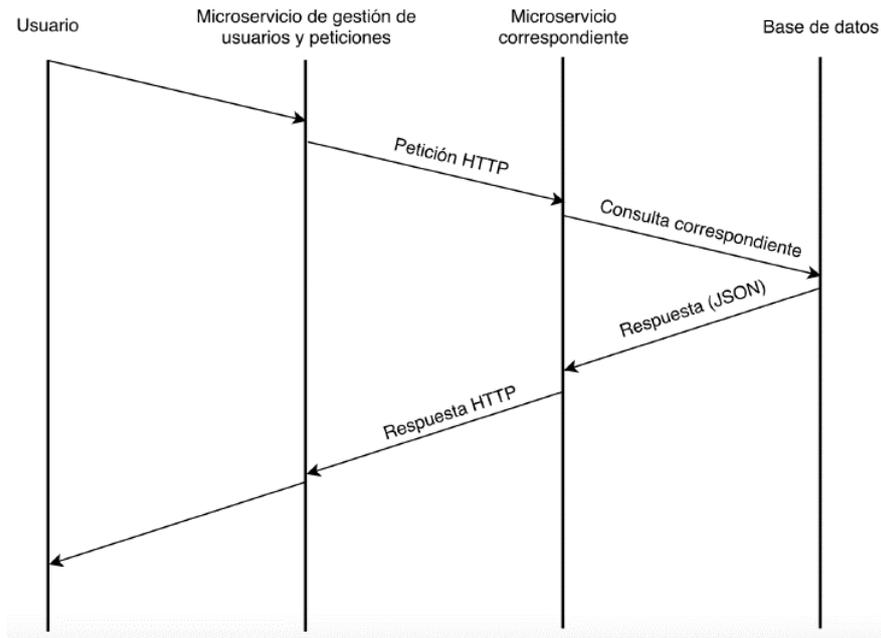


Figura 11. Esquema del funcionamiento interno de las peticiones.  
Fuente: <http://oa.upm.es/37346/>

El esquema que presenta la figura 11 muestra el flujo de funcionamiento interno de los microservicios, empezando por la petición del cliente (frontend) al servidor (backend), el cual envía una solicitud y según la petición se hace la consulta para brindar una repuesta. (Cuadrillero Pérez, 2015)

### Comparativo

MODELO	VENTAJAS	DESVENTAJAS	SIMILITUDES	DIFERENCIAS
Patron: API Gateway/ Backend para frontend	<ul style="list-style-type: none"> <li>Los clientes no se dan cuenta del funcionamiento de los servicios.</li> <li>El cliente no tendrá inconvenientes en la ubicación de los servicios</li> <li>Proporciona acceso a los diferentes clientes.</li> <li>Reduce el número de solicitudes cuando viajan de los datos.</li> <li>Simplifica al cliente cuando se hacen</li> </ul>	<ul style="list-style-type: none"> <li>Mayor complejidad para construir la puerta de enlace API ya que es otra parte de la arquitectura.</li> <li>Mayor tiempo de repuesta según el estado de la red.</li> </ul>	<ul style="list-style-type: none"> <li>Mecanismos estratégicos para comunicar el frontend con el backend.</li> <li>Distribución de servicios web.</li> <li>Uso de tecnologías apropiadas para la construcción de los servicios web.</li> <li>Separación del cliente con el servidor.</li> </ul>	<ul style="list-style-type: none"> <li>Utiliza un API Gateway intermedio para comunicar tanto el frontend con el backend.</li> <li>Api encarga de brindar acceso desde las diferentes plataformas tecnologías (móvil, ordenador, Tablet, etc.)</li> <li>Comunicación entre los servicios web.</li> </ul>

	<p>numerosas peticiones al servidor.</p> <ul style="list-style-type: none"> <li>• Traduce la comunicación un protocolo estándar a cualquier protocolo interno (HTTP).</li> </ul>			
<p>Modelo de integración de tecnologías web para la gestión de contenido virtual B2B</p>	<ul style="list-style-type: none"> <li>• Los clientes no conocerán el funcionamiento interno del modelo del negocio.</li> <li>• Integra las características más importantes de cada tecnología.</li> <li>• El agente decide los filtros entre en backend y frontend.</li> <li>• Manejo de interoperabilidad entre servicios web, agentes y modelado virtual.</li> <li>• Servicios web distribuidos individualmente para los filtros del agente en la solicitud y entrega de información.</li> <li>• Utilización de componentes.</li> <li>• Automatización.</li> <li>• Estandarización.</li> <li>• Control de datos.</li> </ul>	<ul style="list-style-type: none"> <li>• Mayor complejidad para la construcción de agentes.</li> <li>• Base de datos intermedia para decir los filtros del agente.</li> <li>• Utilización de agentes con algoritmos robustos.</li> <li>• No existe comunicación entre los servicios web.</li> <li>• Procesos largos de comunicación entre el cliente y los servicios web.</li> </ul>		<ul style="list-style-type: none"> <li>• Utiliza un Agente intermedio para aplicar filtros entre el frontend y backend.</li> <li>• Portal que hace peticiones a un servidor web para usar los filtros del agente.</li> <li>• Majo de una base de datos intermedia para decidir las peticiones</li> </ul>
<p>Estilo orquestado de la arquitectura basada en microservicios</p>	<ul style="list-style-type: none"> <li>• Esquema de funcionamiento interno de la conexión entre frontend y backend.</li> <li>• Respuestas JSON.</li> <li>• Uso de protocolo HTTP para peticiones y respuestas.</li> <li>• Monitor de eventos que</li> </ul>	<ul style="list-style-type: none"> <li>• Difícil migración de aplicaciones monolíticas.</li> <li>• Comunicación entre los microservicios.</li> <li>• Seguridad en las peticiones.</li> </ul>		<ul style="list-style-type: none"> <li>• Microservicio específico para autorizar usuarios.</li> <li>• Particiones y respuesta autenticadas.</li> <li>• Administración de base de datos distribuidos.</li> </ul>

	<p>consume el servicio requerido por el frontend (estilo orquestado).</p> <ul style="list-style-type: none"> <li>• Gestión de usuarios y peticiones (autenticación y seguridad)</li> <li>• Distribución de microservicios específicos para dar respuesta de autorización a los usuarios.</li> <li>• Descentralización en el manejo de base de datos</li> </ul>			
--	--	--	--	--

*Tabla 10. Análisis de los modelos propuestos para el uso de microservicios.  
Fuente: Autor del proyecto.*

## 4 Diseño de la estrategia de integración de plataformas web dinámicas.

Este capítulo explica la definición de una estrategia que permite la integración de diferentes tecnologías web dinámicas, para la construcción de aplicaciones web; a su vez se detallan los pasos desarrollados de su funcionamiento, lo que permitió la identificación de los elementos que se tomaron en cuenta para la generación de un nuevo enfoque del desarrollo de software web, en base a las características, ventajas y desventajas de su composición.

La estrategia propuesta parte de la concepción de una aplicación web como un conjunto de **funcionalidades públicas o privadas** programadas por dos grandes partes que separan el código de cada funcionalidad: el **frontend** y **backend**.

Las **funcionalidades públicas** pueden ser accedidas por un usuario anónimo, este puede ingresar a la aplicación web solo para consultar información que no requiera ningún tipo de permiso.

En este sentido, las **funcionalidades privadas** solo pueden ser accedidas por usuarios con permisos sobre estas, ya que estos pueden ingresar, modificar, eliminar o buscar información específica de la aplicación web.

El código frontend de cada funcionalidad se desarrolla teniendo en cuenta la tecnología **AngularJS** y **SPA**, es decir que cada funcionalidad se verá como una pequeña aplicación casi totalmente separada de las otras (microservicio) y probablemente compartiendo tan solo la base de datos. Siendo así, el backend de una funcionalidad ya sea pública o privada, puede ser desarrollado en una plataforma diferente con respecto a otra funcionalidad, como se muestra a continuación en la figura 12:

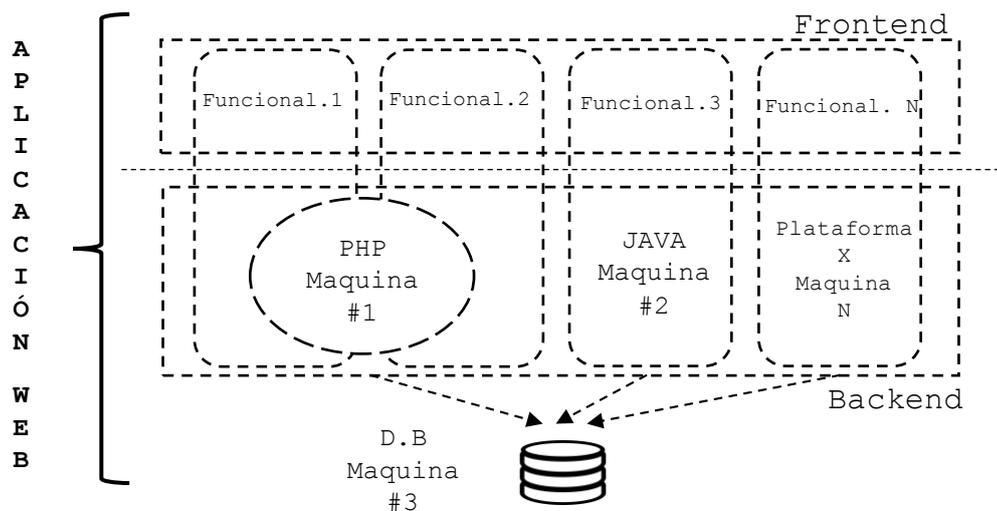


Figura 12. Funcionalidades por plataforma web en una sola aplicación web.  
Fuente: Autor del proyecto.

De igual forma esta estrategia permite ver como una aplicación web puede hacer uso de diferentes máquinas al mismo tiempo, pues algunas funcionalidades se ejecutan en una maquina mientras de manera simultánea otras funcionalidades pueden ser ejecutadas en otras máquinas, lo que puede verse como una forma de paralización del trabajo.

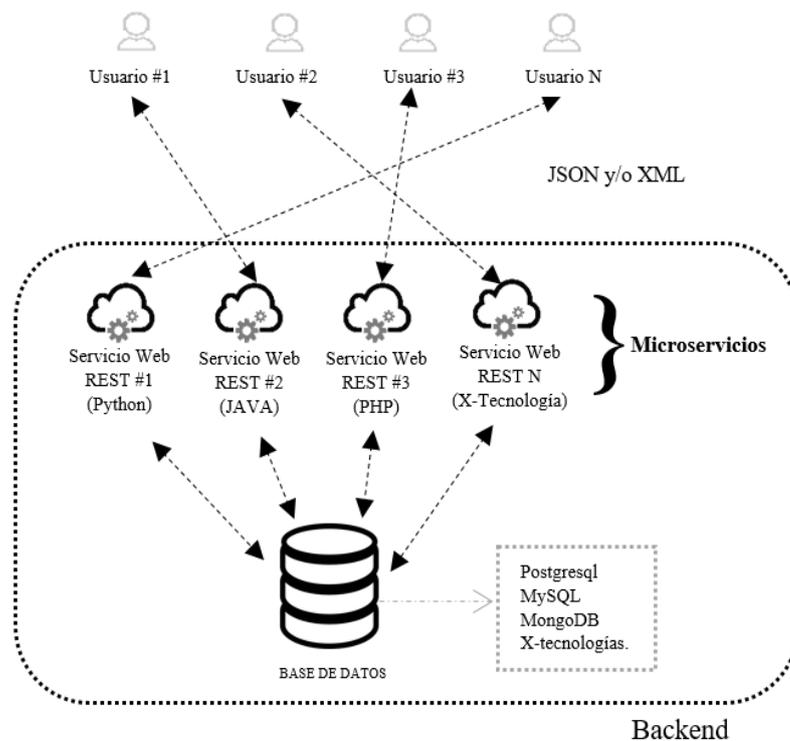


Figura 13. Comunicación de usuarios con funcionalidades.  
Fuente: Autor del proyecto.

Como se puede observar en la figura 13, todos los usuarios conectados a una aplicación web acceden a cada una de las funcionalidades a las cuales tienen permisos o no de manera transparente sin darse cuenta en que máquina está ejecutando los servicios desarrollados mediante su código backend.

Típicamente estas máquinas estarán diferenciadas por la plataforma que soportan como puede ser PHP, JSP, Python, entre cualquier otra tecnología del lado del servidor en una aplicación web típica como se indica en la figura 13.

Esta tecnología permite que el procesamiento de datos se haga en forma paralela entre funcionalidades de una misma aplicación.

Cabe agregar el funcionamiento interno entre el frontend y el backend, el cual se basa en un flujo interno de comunicación entre ambas partes, por lo tanto, cuando un usuario envía una solicitud desde el frontend este lo redirige al backend, dicha solicitud es captada por la parte lógica del negocio para luego ser operada en la base de datos y retornarla al usuario, como se puede observar en la figura 14.

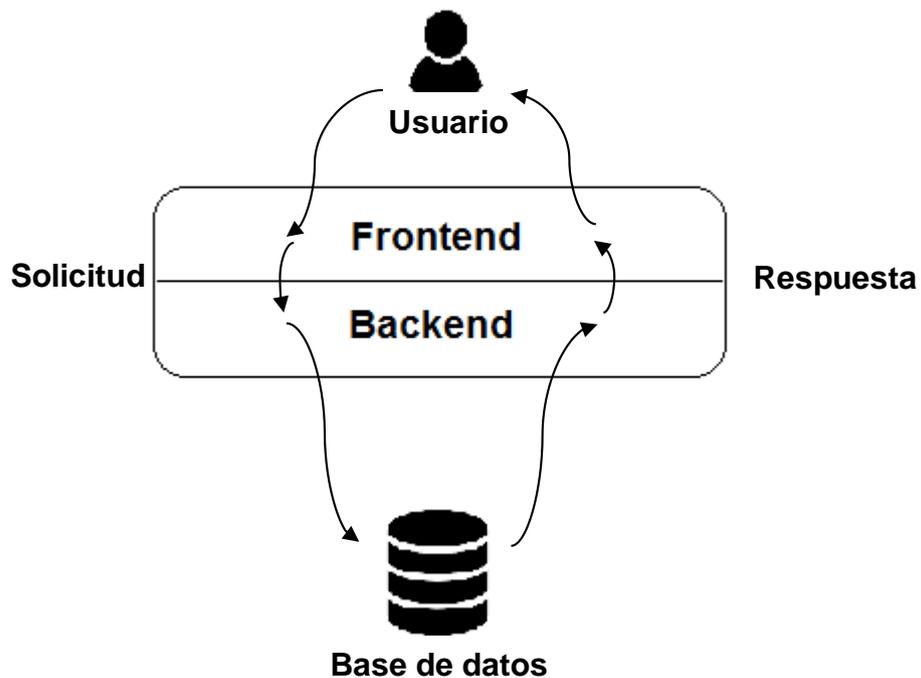


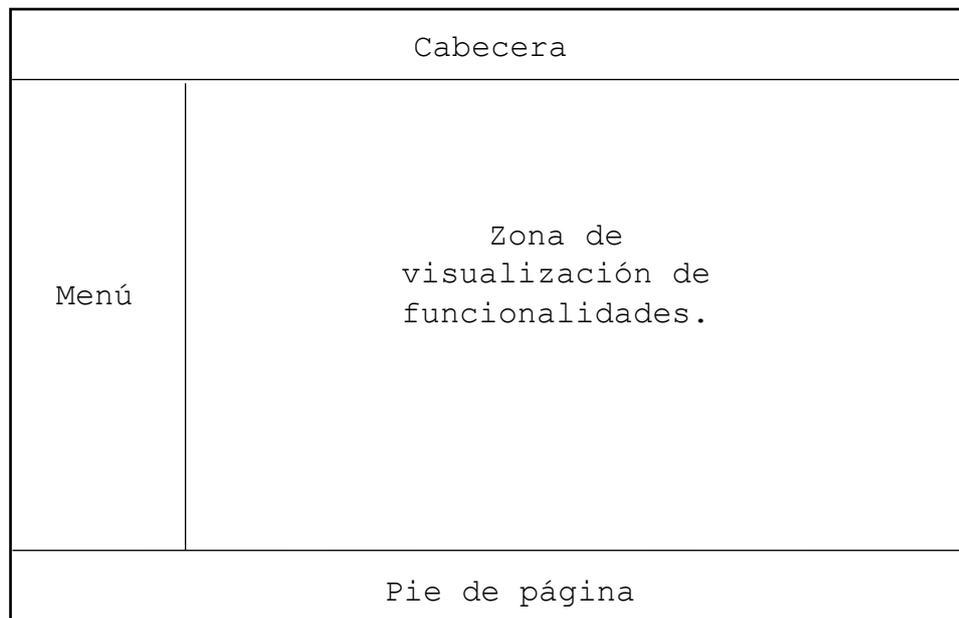
Figura 14. Flujo básico entre el frontend y el backend.  
Fuente: Autor del proyecto.

El **frontend** de cada funcionalidad estará desarrollado bajo la tecnología **AngularJS** de acuerdo al modelo vista controlador, el cual contendrá toda la parte interactiva y vistas del usuario de cada funcionalidad de la aplicación web basada en una única página web (**SPA**).

El **Backend** de cada funcionalidad contendrá toda la parte lógica del negocio, la cual será desarrollado bajo la arquitectura **REST** para comunicar el cliente con el servidor de acuerdo a las diferentes solicitudes que se hagan por parte el usuario, por lo tanto, dichas solicitudes serán invocadas y consumidas bajo los métodos del **protocolo HTTP** (POST, GET, DELETE, PUT, ...), cada servicio web REST (**PHP, JAVA, Python, ...**) estará dividido de las otras plataformas para construir los recursos necesarios de acuerdo a la funcionalidad que vaya acceder el usuario.

De esta forma la aplicación web **SPA** se va diseñar en base a la estructura de la figura 15, el objetivo de este diseño es definir una estructura estándar de una página web que estará dividida en varias zonas.

El propósito de este diseño se basa en la construcción de una única página web dinámica (SPA) que contiene varias zonas para estructurar los contenidos visualizados por el usuario, estos contenidos serán controlados por un menú, el cual es el encargado de brindarle al usuario todas las funcionalidades, ya sean públicas o privadas, y desplegarlas en la zona de visualización de funcionalidades para una navegación óptima sin tener que recargar la página web cada vez que se haga una operación en ella.



*Figura 15. Estructura de una aplicación web SPA.  
Fuente: Autor del proyecto.*

En términos generales, la estrategia se enfocará en una única página web para desplegar la información dinámicamente sin tener que cargar y viajar a páginas adyacentes que puedan congestionar la navegación, de esta manera se le va

brindar al usuario una experiencia más fluida en los procesos que este haga para optimizar una navegación más ligera.

Un detalle importante a destacar en la estrategia propuesta es que se requiere de una funcionalidad de ingreso para las funcionalidades privadas de la aplicación web, ya que esta permite acceder a los usuarios que tienen roles asignados para manipular la información, pues las funcionalidades públicas solo puede hacer consultas sin alterar la información.

### Control de acceso a la aplicación web

La estrategia propuesta incluye el registro y administración de la información de la aplicación web respecto a usuarios organizados en roles, módulos de aplicación, funcionalidades de cada módulo y permiso de acceso de cada rol a las funcionalidades privadas, como se puede observar en la figura 16. Esta información estará en la base datos de la misma aplicación en un conjunto de tablas predefinidas.

De este modo, los usuarios que no tengan ningún permiso (usuarios anónimos) solo podrán visualizar o consultar el contenido de las funcionalidades públicas, mas no administrarlo (eliminar, actualizar), como se puede observar en la figura 16.

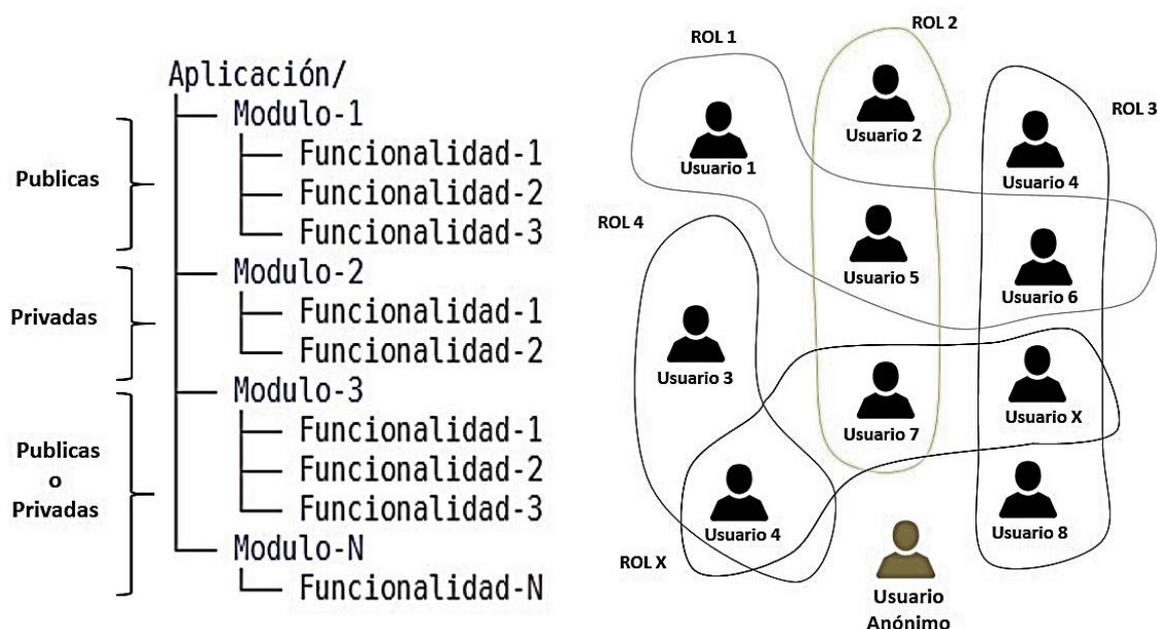


Figura 16. Estructura de la aplicación web por acceso.  
Fuente: Autor del proyecto.

En este sentido las funcionalidades que sean privadas funcionaran de acuerdo a los clientes que ingresen a la aplicación web a través de una solicitud de

autorización, la cual será la encargada de brindarle los privilegios de acceso al sistema web.

En relación a lo anterior, cuando el cliente ingresa por primera vez, este se conecta a la maquina principal, que será la encargada de brindarle y asignarle todas las funcionalidades privadas que tiene asociadas según el rol en la aplicación web. Cuando se elija una funcionalidad, esta se encuentra alojada en una maquina separada de la principal, la cual se conecta a los datos y procesa la información en el backend.

Del mismo modo, se devuelve al frontend, formando un ciclo repetitivo por cada funcionalidad que se trabaje entre el frontend y backend de la maquina encargada de brindarle el servicio, de igual forma funcionara para los clientes que no estén autorizados, con la única diferencia de que solo podrá visualizar contenido que no esté restringido.

En la figura 17 se puede apreciar el flujo básico del cliente con sus funciones las cuales se ejecutan en distintas maquinas con diferentes procesos en la información que sea requerida por el cliente para que todo se integre en una sola aplicación web, pero trabajando de forma independiente.

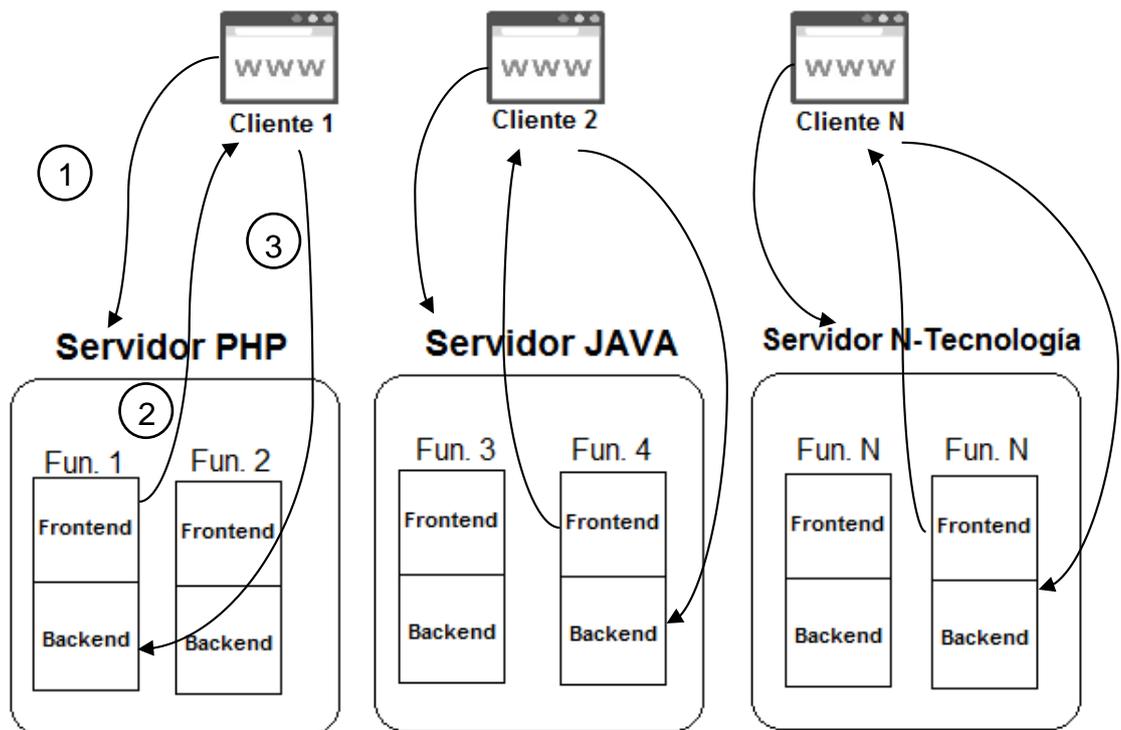


Figura 17. Acceso del cliente a las funcionalidades.  
Fuente: Autor del proyecto.

## Almacenamiento de código

Tanto el código del backend como el frontend de cada funcionalidad estará almacenado en la máquina que soporte la tecnología para la cual se diseñó dicha funcionalidad: PHP, JSP, Python, etc.

## Proceso de login para acceder a las funcionalidades privadas

El proceso de login en una aplicación web que utilice esta estrategia, se constituye en la puerta de entrada a la aplicación, por lo tanto, se requiere que este código este almacenado en la máquina que soporte esta tecnología, y se constituye como la maquina principal a la cual se debe acceder por primera vez para poder administrar las funcionalidades privadas.

Es importante destacar el flujo de acceso a la aplicación web en ejecución, para ello se determinó un control de ingreso por usuario, el cual se basa en dos casos muy importantes dependiendo de los roles este tenga asignados.

El primer caso consiste en el flujo de acceso para el usuario que tenga asignado más de un rol, por lo tanto, este usuario debe elegir el rol y posteriormente se le desplegará un menú en forma de árbol que cargara dinámicamente las funcionalidades que no son públicas y tenga asignadas, como se observar en el punto uno de la figura 18.

De igual forma el segundo caso se da cuando el flujo de acceso para el usuario que solo tenga asignado un solo rol, el cual entrará a la aplicación web directamente con las funcionalidades privadas propias de este rol, como se puede apreciar en el punto dos de la figura 18.

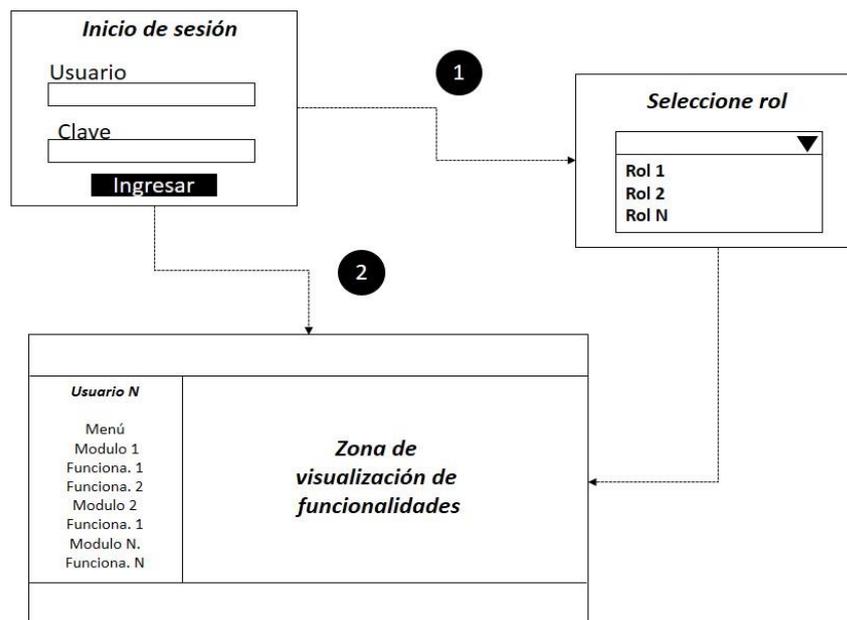


Figura 18. Estructura de acceso a la aplicación web por usuario.  
Fuente: Autor del proyecto.

## 4.1 Aspectos técnicos

Por otra parte, con el fin de construir la aplicación web con respecto a la estrategia se debe tener en cuenta aspectos técnicos que faciliten la comunicación y seguridad en las peticiones entre el **frontend** y el **backend**, por lo tanto, se debe tener en cuenta lo siguiente:

- **Comunicación entre frontend y el backend**

Es importante destacar la comunicación del portal web con los pequeños servicios web (microservicios) que se encuentran distribuidos en diferentes máquinas, esta se hará con la ayuda del framework AngularJS, por lo que este framework permite una comunicación HTTP asíncrona para tener un control en ambos extremos de la comunicación, otro aspecto para resaltar en este apartado se basa en brindar una mínima seguridad en ambas direcciones cuando viaja información del frontend al backend y viceversa, en las diversas peticiones que se hagan por parte del usuario en relación a los datos para enviar y obtener información de acuerdo a la funcionalidad privada que se esté ejecutando.

Posteriormente se sugiere implementar los **protocolos HTTP** de comunicación existentes como por ejemplo entre los más comunes y que se han mencionado en capítulos anteriores, algunos de estos métodos son **POST, GET, DELETE, PUT,...**, los cuales serán los encargados de hacer las peticiones del cliente al servidor a través de una ruta de acceso (**URL**), esta ruta será la encargada de hacer la solicitud de dicha consulta requerida por el usuario, cuando la consulta es enviada, esta tendrá un **dirección IP** para comunicar con una máquina específica, la cual tendrá alojado el servicio web (**API**).

De este modo es importante destacar los códigos de estados de la respuesta del **protocolo HTTP** de acuerdo a la solicitud que se haga en backend, donde el servicio web responderá con un estado **HTTP**, estos indican si se ha completado satisfactoriamente la solicitud al servicio web, dependiendo de la respuesta que este emita se invocarán los siguientes estados (mensajes); estado 200-226 que indica peticiones correctas, 300-308 redirecciones, 400-451 errores del cliente, 500-511 errores del servidor, esto varía dependiendo de la respuesta que devuelva el servidor web.

Asimismo, cuando la respuesta es correcta los datos se guardarán un espacio en memoria creado por el framework **AngularJS** ya sea en vectores, Arrays, variables, entre otros métodos propios de **AngularJS**, donde solo contendrá texto ligero ya sea en formato **JSON** o **XML** el cual será formateado por las diferentes funciones que contenga el framework para luego retornar de manera clara y precisa la información a la funcionalidad privada que fue solicitada por el usuario.

- **Acceso a las funcionalidades por usuario.**

El acceso a las funcionalidades es de vital importancia en la aplicación web, ya que estas se dividen en dos partes; **públicas** o **privadas**, de este punto de vistas, las funcionalidades públicas son de libre acceso (usuarios sin credenciales), en cambio las privadas necesitan algún tipo de autenticación para ingresar a visualizar su contenido (usuarios con credenciales).

- **Usuarios con credenciales (funcionalidades privadas)**

El propósito de brindar una mínima seguridad en ambos extremos de la comunicación se enfocará en proteger parte de la información cuando esta viaja entre el **frontend** y el **backend** o viceversa, para ello se debe implementar como básico un control de usuarios y roles, estos estarán registrados en la base de datos, los registros serán consultados por la funcionalidad de usuarios, la cual es la encargada de controlar todos los usuarios para desplegar sus módulos y funciones que pueden operar en la aplicación web, dependiendo de las funcionalidades privadas que tengan asignadas y permisos que este tenga asociados en un conjunto de roles estipulados en la base de datos.

En relación a lo anterior se debe implementar un login para ingresar a las funcionalidades privadas que contiene la aplicación web, a diferencia de los métodos de sesión de las aplicaciones web tradicionales, se va a utilizar **tokens** entre la comunicación para proponer una mínima seguridad en ambos extremos a través de un filtro ubicado en cada uno de los microservicios, que validara la autenticación del usuario tanto en **frontend** como del **backend** cuando viaje información, pero esto no quiere decir que se va ofrecer una seguridad completa en la aplicación web que se desarrolle.

En este sentido el uso de los **tokens** facilitará el control interno en la comunicación en ambas direcciones de la aplicación web para controlar las peticiones y respuestas del servidor hacia el usuario, por ello se establece generar diferentes **tokens** a los usuarios cuando estos se conecten a la aplicación web, por lo tanto este será único para cada usuario, este a su vez se almacenará en el navegador web para ser reutilizado las veces que sea necesarias en las peticiones del usuario de forma transparente.

○ **Usuarios sin credenciales (funcionalidades públicas)**

Los usuarios que no se encuentren registrados en la base de datos, se verán como usuarios anónimos de la aplicación web, estos solo podrán visualizar o consultar información sin ningún tipo de autenticación, con la limitante de que no podrán modificar contenido de las funcionalidades privadas.

Esto indica que los usuarios anónimos no tendrán la opción de solicitar acceso a ciertos recursos (funcionalidades privadas) de la aplicación web, pues estos usuarios no pasaran por ningún filtro de autenticación, si no por lo contrario solo ingresar a las funcionalidades públicas directamente, sin ningún permiso para acceder a ellas.

Finalmente, su acceso en la aplicación web, no incluirá ningún tipo de autorización, pues será libre su acceso solo a las funcionalidades públicas.

## **4.2 Características, ventajas y desventajas.**

Más allá de la definición, para poder entender mejor esta estrategia, hay que analizar sus principales características, ventajas y desventajas.

### **Características**

- Separación del cliente con el servidor.
- Arquitectura MVC (vista y controlador en el cliente y modelo en el servidor).
- Uso de microservicios.
- Uso de framework del lado del cliente.
- Implementación de la arquitectura REST.
- Integración de plataformas web (JSP, PHP, Python, entre otras).
- Portar web de una sola página (SPA).

- Uso de APIs.
- Bases de datos.
- Máquinas (servidores) trabajando de forma independiente.
- Implementación de tokens para ofrecer una mínima seguridad en la comunicación cuando viajan los datos.

### **Ventajas**

- Mejorar el proceso de rendimiento en el backend ya que los servicios que se ofrecerán al usuario trabajarán de forma independiente.
- Se podrá utilizar cualquier lenguaje de programación en los diferentes servicios web (microservicios) creados.
- A diferencia de las aplicaciones web tradicionales, esta estrategia hará uso de SPA (aplicación de una sola página) sin tener que viajar a páginas adyacentes.
- Implementación de tokens como medio de seguridad en la comunicación del **frontend** con el **backend** o viceversa.
- Facilitará la construcción del **frontend**, ya que este contará con la implementación de un framework tanto para el desarrollo como para el diseño.
- Servicios web independientes (microservicios).
- Uso de las nuevas arquitecturas de desarrollo web (REST).

### **Desventajas**

- Construcción un poco pausada en cada servicio web (microservicio) ya que esta parte no contará con la ayuda de un framework de desarrollo.
- Fallos de seguridad cuando viajen los datos de extremo a extremo, ya que los tokens no cubren la totalidad de seguridad de la información.

### **4.3 Validación del prototipo integrasoft**

Esta sección hace referencia a la validación de la estrategia propuesta anteriormente, por lo tanto, se implementó un prototipo que validará el funcionamiento de esta, para determinar su eficiencia en la integración de plataformas web dinámicas (Java, PHP, Python, etc....).

#### **4.3.1 Descripción del prototipo integrasoft.**

La administración de cualquier aplicación web requiere de la definición de los servicios (funcionalidades) que ofrece la aplicación a los usuarios que acceden y los permisos que estos tienen sobre dichos servicios.

En este sentido se implementaron las funcionalidades privadas para la gestión de usuarios, gestión de roles, gestión de funcionalidades y login, las cuales serán las encargadas del control básico de una aplicación web.

La funcionalidad gestión de usuarios es la encargada de contener los datos de las personas y usuarios de la aplicación web.

De igual forma la funcionalidad gestión de roles es la encargada de contener todos los roles que puede tener un usuario en la aplicación web, ya que de aquí parten todos los privilegios o permisos que los usuarios tiene para realizar distintas actividades sobre el procesamiento de los datos que se manipulan en la aplicación web.

Posteriormente la funcionalidad gestión de funcionalidades es la encargada de contener todos los módulos encargados de las actividades que se puede ejecutar sobre la aplicación web, estos módulos solo pueden ser accedidos por los roles que tengan permisos sobre ellos.

De este modo la funcionalidad login es la encargada de gestionar y controlar todos los usuarios registrados en la aplicación web, esta funcionalidad también se encarga de verificar los roles que tienen los usuarios para determinar a qué módulos puede acceder.

En base a lo anteriormente descrito, se elaboró el modelo de la base de datos de acuerdo a la figura 19.

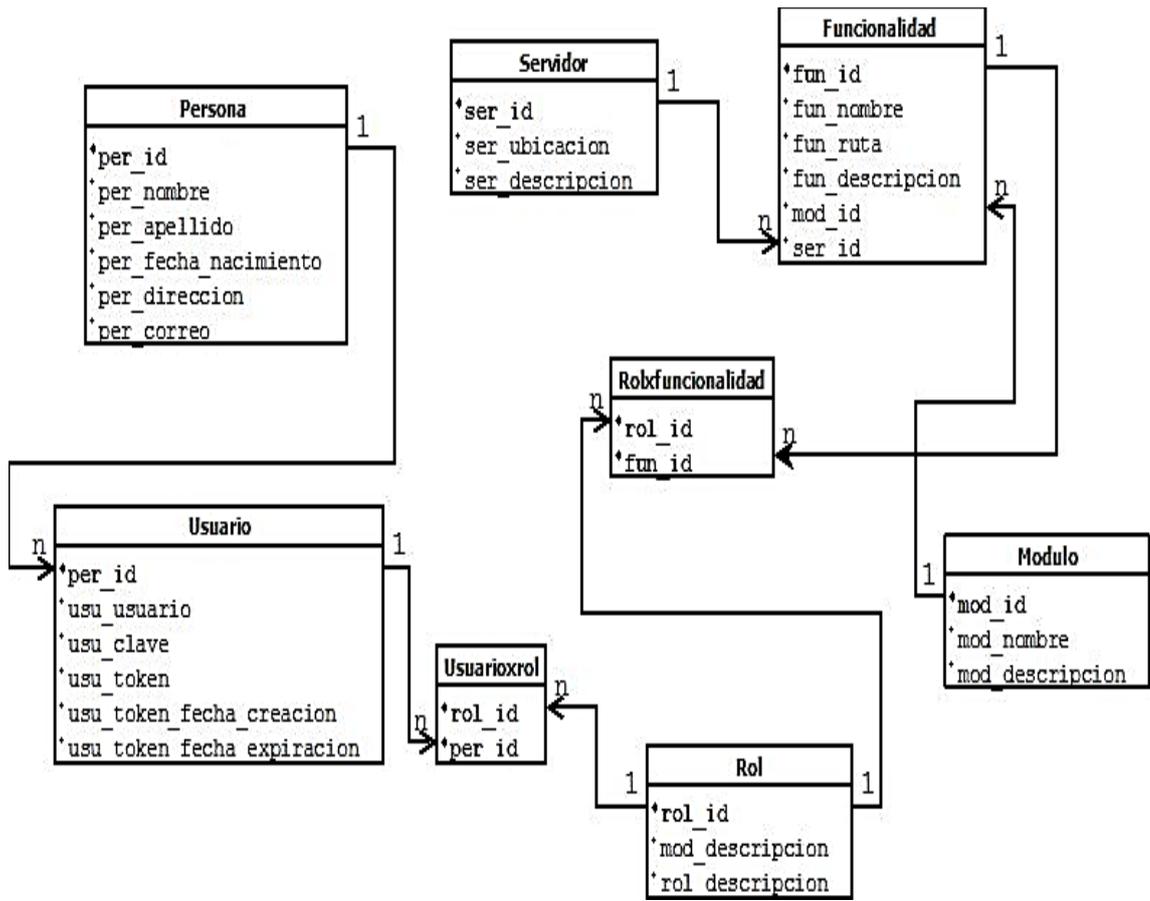


Figura 19. Modelo entidad – relación (MER) de la base de datos.  
Fuente: Autor del proyecto.

### 4.3.2 Funcionamiento interno del prototipo

En términos generales el prototipo se llevó a cabo con la implementación de un portal web y dos microservicios alojados en diferentes máquinas, el portal web fue construido bajo el framework de desarrollo **AngularJS**, debido a sus ventajas propias que éste tiene en cuanto al ahorro de código y tiempo a la hora de construir software web del lado del cliente (frontend), por consiguiente se creó una arquitectura **MVC** para las diferentes funcionalidades privadas de roles, usuarios y módulos bajo la arquitectura **RESTful**, estas se construyeron en base a las tecnologías web de **PHP** y **JSP**, cada una de ellas fue utilizada para elaborar los diferentes servicios web divididos en varias máquinas que brindan funciones diferentes a las peticiones del usuario.

Por otra parte, cada funcionalidad consta de un controlador y sus correspondientes vistas en el **frontend** para cada usuario, las funcionalidades tendrán un **index.js** que funciona como controlador de todas las peticiones que se hagan a los microservicios web, para enviar las solicitudes a las vistas del usuario y desplegar todas las peticiones que fueron recurridas por este.

Asimismo, cada controlador hará uso del **protocolo HTTP** como canal de comunicación entre en **frontend** y el **backend**, con el uso de los métodos **POST**, **PUT**, **GET**, **DELETE**, entre otros; Las vistas serán controladas por el archivo **index.js** correspondiente en cada funcionalidad, en las vistas se usó código **HTML** y **JavaScript** para ofrecerle una estructura amigable al usuario.

De esta forma los dos microservicios que se desarrollaron usaron las tecnologías **JAVA** y **PHP**, de modo que los servicios web se implementaron bajo la **arquitectura REST** como **APIs** independientes alojadas en diferentes máquinas, cada microservicio está organizado por un modelo que es el encargado de recibir las peticiones del portal web para realizar la operación **CRUD** correspondiente en la base de datos, es de notar que existe un intermediario entre el modelo y la base de datos que hace el rol de **DAO** (objeto de acceso a datos).

En este sentido cada microservicio contiene una conexión a la base de datos, objetos de acceso a los datos (**DAO**) y los modelos correspondientes para cada funcionalidad, las cuales fueron divididas y ubicadas en los servicios web **PHP** y **JAVA** en máquinas independientes.

### **4.3.3 Funcionalidades implementadas en el prototipo.**

Las funcionalidades que fueron implementadas en el prototipo, serán las encargadas de validar lo propuesto por la estrategia anteriormente descrita, para ello se implementaron las siguientes funcionalidades; login, gestión de usuario, gestión de funcionalidades, gestión de roles.

#### **4.3.3.1 Login**

Esta funcionalidad se llevó a cabo para tener un control detallado de todos los usuarios que tienen acceso a la aplicación web, debido a que cada uno de ellos puede contar con uno o varios permisos a la vez, por esta razón se realizó un acceso a la aplicación web, ya que existen diversos roles para operar sobre ella.

De este modo la funcionalidad login consta de una serie de gestiones, estas se van ejecutando cada vez que el usuario haga un nuevo inicio de sesión; para empezar es necesario que el usuario digite sus credenciales, las cuales serán únicas para cada usuario que va ingresar a la aplicación web, luego se listarán el rol(s) que tiene asignados, cuando se elija alguno de ellos, este tendrá acceso a todas las funcionalidades que tenga permisos, las cuales se ejecutarán en una zona específica de la página web.

En relación a lo anterior, la implementación de esta funcionalidad, se construyó bajo la tecnología **PHP**, este fue implementado por su código abierto, flexible y potente para desarrollar aplicaciones web.

- **Tecnologías usadas**

- **Bootstrap** como framework de diseño para las vistas del usuario.
- **AngularJS** como framework de desarrollo en el frontend.
- **PHP** para crear el microservicio de la funcionalidad login en el backend.
- **PostgreSQL** como motor de base de datos.

- **Diseño del Storyboard de la funcionalidad login.**

En cuanto al funcionamiento externo de la funcionalidad login, esta se construyó en base al Storyboard de la figura 20, la cual indica el proceso de las vistas de esta funcionalidad, esta parte de la primera interacción del usuario con la aplicación web, empezando por el ingreso del usuario, hasta su salida, esto estará ubicado en la pantalla principal de la aplicación web, la cual va ser única y semejante para los demás usuarios.

De esta forma el usuario tendrá un menú dinámico de acuerdo al rol que haya elegido o tenga asignando (usuario con un solo rol).

Posteriormente el usuario deberá elegir la funcionalidad que va trabajar, la cual se le desplegará en la zona de visualización de funcionalidades, esta zona le mostrará al usuario las diversas operaciones que puede ejecutar sobre ella.

Finalmente, cuando el usuario haya terminado todas sus operaciones en la aplicación web, este tiene la opción de salir, la cual tendrá una confirmación de salida para terminar todos sus procesos.

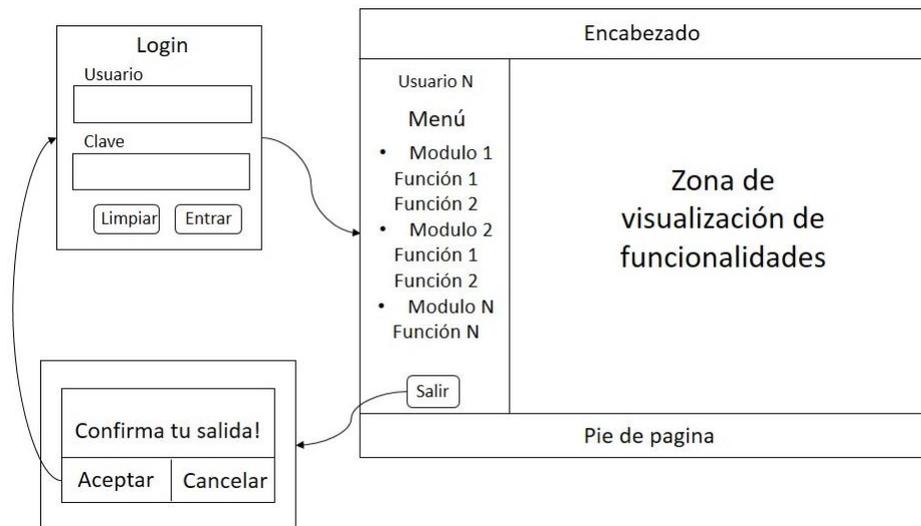


Figura 20. Storyboard de funcionalidad login.  
Fuente: Autor del proyecto

- **Diagrama de secuencia entre frontend y el backend de la funcionalidad login.**

Es importante destacar la comunicación entre el frontend y el backend, ya que cada uno de ellos opera de forma independiente a la otras en diferentes maquinas, para ello se llevó a cabo el diagrama de la figura 21, el cual describe el funcionamiento básico de cada una de las funcionalidades.

En este sentido la funcionalidad login está estructurada con un inicio de sesión en el frontend, este será el encargado de solicitar al usuario sus credenciales para entrar a la aplicación web y determinar si el usuario es válido o no.

De igual forma, esta funcionalidad cuenta con una función que validará el usuario en el backend, con la ejecución del microservicio encargado de consultar los datos del usuario en la base de datos y brindarle una respuesta al frontend, el cual decidirá su ingreso a la aplicación web.

Como consecuencia a lo anteriormente descrito, la conexión entre el frontend y backend, se llevó a cabo en base a la figura 21.

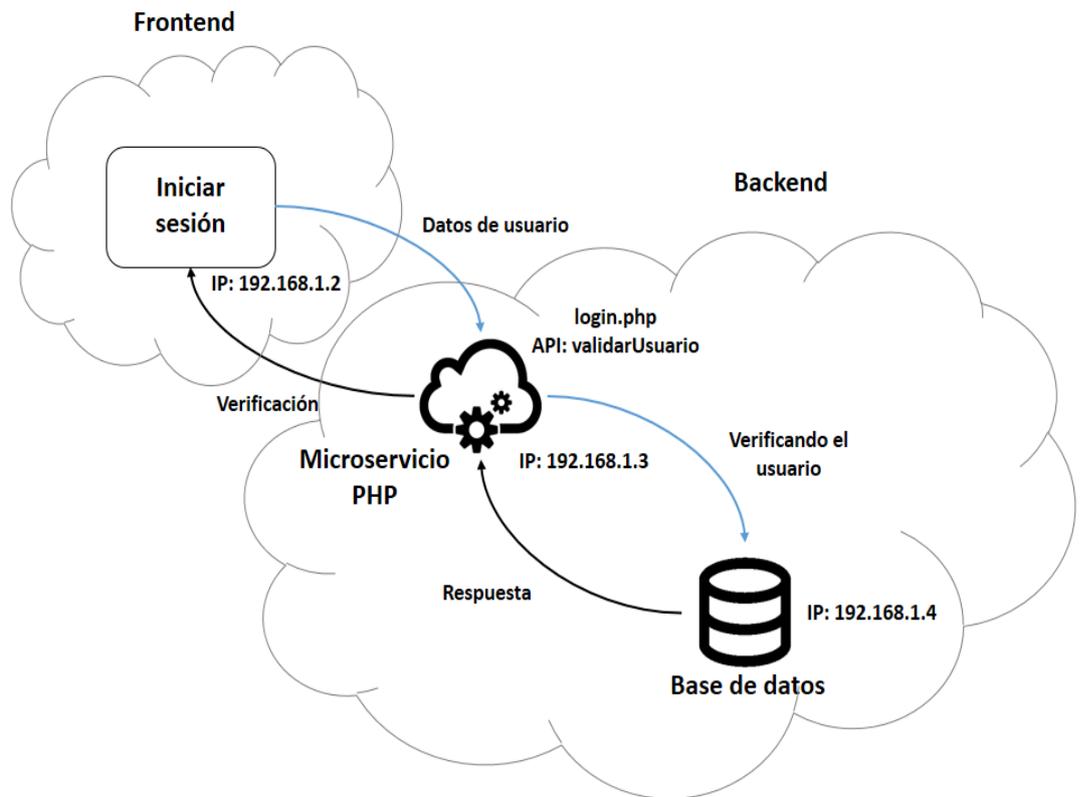


Figura 21. Diagrama de secuencia de la funcionalidad login.  
Fuente: Autor del proyecto.

- **Código implementado**

En base a lo anteriormente planteado, se implementó el código de la figura 22 para funcionalidad login en el frontend desarrollado bajo la tecnología **AngularJS**.

De esta manera, este código es el encargado de enviar los datos, en esta ocasión las credenciales del usuario al microservicio login, mediante una URL, la cual contiene la dirección IP de la máquina que ofrece este servicio.

De este modo, el microservicio envía la respuesta de la solicitud del frontend, este la recibe y ejecuta las acciones correspondientes para que el usuario pueda ingresar o no a la aplicación web.

```

$scope.submit = function (dato) {
    var json = {usu_usuario: dato.usu_usuario, usu_clave:
    dato.usu_clave};
    $http({
        method: "POST",

```

```

        headers: {'Accept': 'application/json, text/plain',
        'Authorization': ''},
        url: APIURLPHP + "/modeloApi/login/buscar",
        data: JSON.stringify(json)
    }).then(function (response) {
        $scope.datos = response.data;
        console.log(response);
        if ($scope.datos.msg === 'true') {
            sesionControl.set('estado', $scope.datos.msg);
            sesionControl.set('token', $scope.datos.token);
            sesionControl.set('id', $scope.datos.id);
            console.log(response);
            window.location.replace('login');
        } else {
            $scope.error = "Usuario y/o password incorrectos...";
        }
    }, function (statusText) {
        console.log(statusText);
    });
};

```

Figura 22. Método para ingresar a la aplicación web.

Fuente: Autor del proyecto.

Este método es el encargado de recibir y conectar con el objeto de acceso a los datos (DAO), para darle la respuesta a la solicitud del frontend, como se indica en la figura 23.

```

<?php
class loginAPI {

    private $dao;

    function __construct() {
        $this->dao = new loginDAO();
    }

    public function API() {
        $method = $_SERVER['REQUEST_METHOD'];
        switch ($method) {
            case 'GET':
                break;
            case 'POST':
                switch ($_GET['ejecutar']) {
                    case 'buscar':
                        $datos = $this->dao->buscarUsuario(
                            file_get_contents("php://input"));
                        echo json_encode($datos);
                        break;
                    case 'listar':
                        $datos = array("datos" => $this->dao->
                            listarFuncionalidad(file_get_contents("p
                            hp://input")));
                        echo json_encode($datos);
                        break;
                    default :
                        echo 'Ruta sin acceso';
                }
            }
        }
    }
}

```

```

        }
        break;
    case 'PUT':
        break;
    case 'DELETE':
        break;
    default:
        echo 'no encontro metodo';
        break;
    }
}
}
}

```

Figura 23. API de la funcionalidad login.  
Fuente: Autor del proyecto

Los métodos de la figura 24, son los encargados de hacer la consulta a la base de datos.

```

public function buscarUsuario($datos) {
    $index = 1;
    $obj = (array) json_decode($datos);
    $query = "select * "
        . "from usuario as us, "
        . "persona as per "
        . "where us.per_id=per.per_id "
        . "and us.usu_usuario=? "
        . "and us.usu_clave=? ";
    $resultado = $this->conexion->getConexion()-
    >prepare($query);
    $resultado->bindParam($index++, $obj['usu_usuario']);
    $resultado->bindParam($index++, $obj['usu_clave']);
    $resultado->execute();
    if ($resultado->rowCount() > 0) {
        while ($result = $resultado->fetch(PDO::FETCH_ASSOC))
        {
            $per_id = $result['per_id'];
        }
        $header = json_encode(['typ' => 'JWT', 'alg' =>
        'HS256']);
        $payload = json_encode(['usu_usuario' =>
        $obj['usu_usuario'], 'usu_clave' =>
        $obj['usu_clave']]);
        $base64UrlHeader = str_replace(['+', '/', '='], ['-',
        '_', ''], base64_encode($header));
        $base64UrlPayload = str_replace(['+', '/', '='], ['-',
        '_', ''], base64_encode($payload));
        $signature = hash_hmac('sha256', $base64UrlHeader .
        "." . $base64UrlPayload, 'apiToken', true);
        $base64UrlSignature = str_replace(['+', '/', '='],
        ['-', '_', ''], base64_encode($signature));
        $jwt = $base64UrlHeader . "." . $base64UrlPayload .
        "." . $base64UrlSignature;
        $json = $this->verificarToken($per_id, $jwt);
    } else {
        $json = array("msg" => "false");
    }
}

```

```

        $this->conexion->getCerrarConexion();
        return $json;
    }

    public function verificarUsuario($datos, $headers) {
        $index = 1;
        if (!is_array($datos)) {
            $obj = (array) json_decode($datos);
        }

        $query ="select * "
            ."from usuario as u, "
            ."persona as p, "
            ."usuarioxel as ur, "
            ."rol as r, "
            ."rolxfuncionalidad as rf, "
            ."funcionalidad as f "
            ."where u.per_id=p.per_id "
            ."and u.per_id=ur.per_id "
            ."and ur.rol_id=r.rol_id "
            ."and rf.rol_id=r.rol_id "
            ."and rf.fun_id=f.fun_id "
            ."and fun_nombre=? "
            ."and fun_tipo=true "
            ."and u.per_id=? "
            ."and u.usu_token=? "
            ."and u.usu_token_fecha_expiracion>now() "
            ."order by u.usu_token_fecha_creacion desc "
            ."limit 1";

        $resultado = $this->conexion->getConnection()-
        >prepare($query);

        if (!empty($obj)) {
            $resultado->bindParam($index++,
            $obj['funcionalidad']);
            $resultado->bindParam($index++, $obj['id']);
            $resultado->bindParam($index++, $headers);
            $resultado->execute();
        } else {
            $resultado->execute(array(
                $datos['funcionalidad'],
                $datos['id'],
                $headers
            ));
        }

        if ($resultado->rowCount() > 0) {
            $json = TRUE;
        } else {
            $json = FALSE;
        }
        $this->conexion->getCerrarConexion();
        return $json;
    }
}

```

Figura 24. Métodos de acceso a los datos (DAO).  
Fuente: Autor del proyecto.

### 4.3.3.2 Gestión de usuarios

Esta funcionalidad se llevó a cabo para tener un control detallado de todos los usuarios de la aplicación web.

De este modo la funcionalidad gestión de usuarios es la encargada de registrar usuarios nuevos, eliminar usuario, actualizar usuario, buscar o listar usuarios y gestionar los roles que tiene asignados o por asignar.

- **Tecnologías usadas**

- **Bootstrap** como framework de diseño para las vistas del usuario.
- **AngularJS** como framework de desarrollo en el frontend.
- **JAVA** para crear el microservicio de la funcionalidad gestión de usuarios en el backend.
- **PostgreSQL** como motor de base de datos.

- **Diseño del Storyboard de la funcionalidad gestión de usuarios.**

El funcionamiento externo de la funcionalidad gestión de usuarios, se desarrolló en base a los Storyboards de las figuras que se observan a continuación; las cuales indican el proceso de las vistas de esta funcionalidad.

De esta forma esta funcionalidad es la encargada de gestionar todos los usuarios de la aplicación web, partiendo de registros de usuario, actualización de usuario, eliminación de usuario, buscar o listar usuarios y roles por asignar.

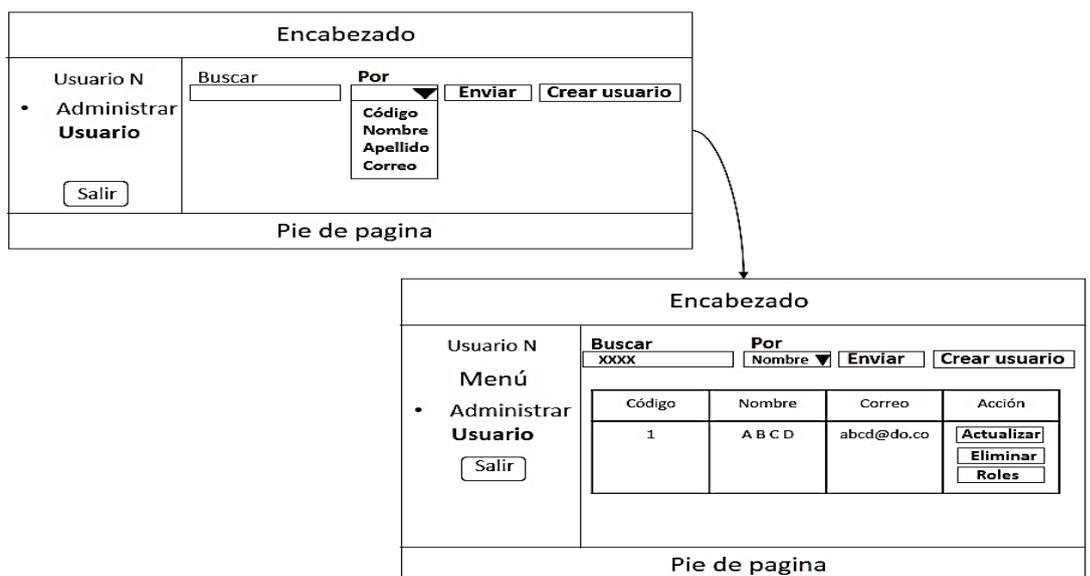


Figura 25. Storyboard buscar usuario.

Fuente: Autor del proyecto.

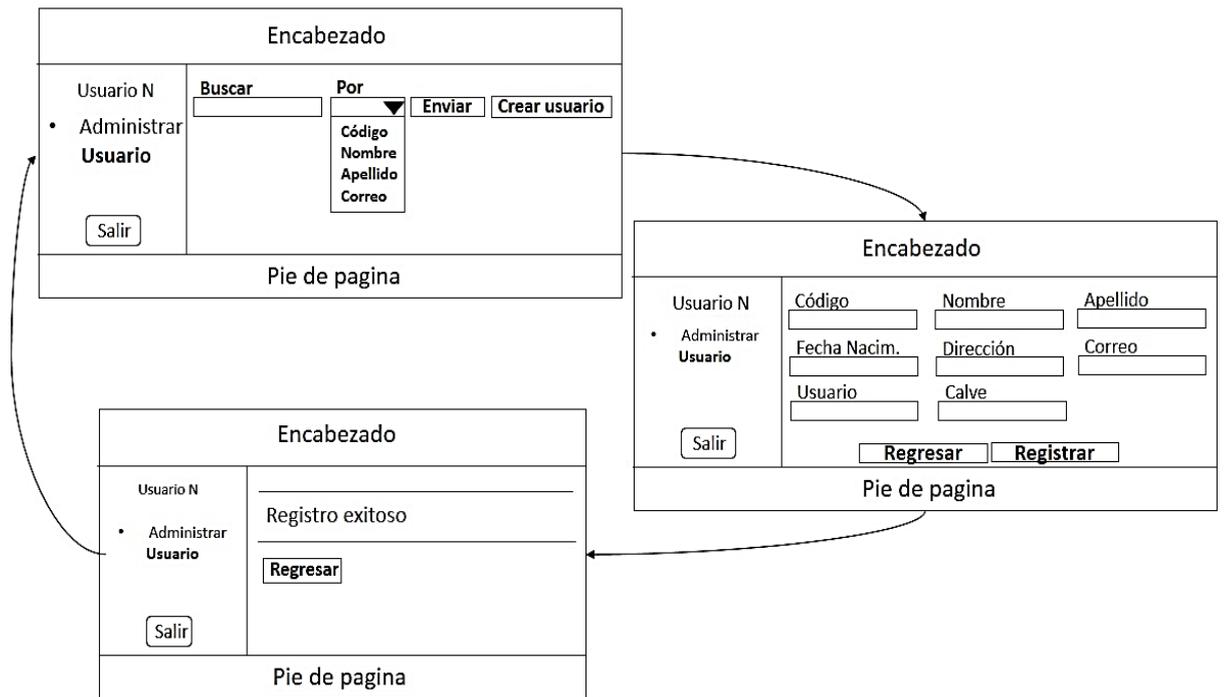


Figura 26. Storyboard crear usuario.  
Fuente: Autor del proyecto.

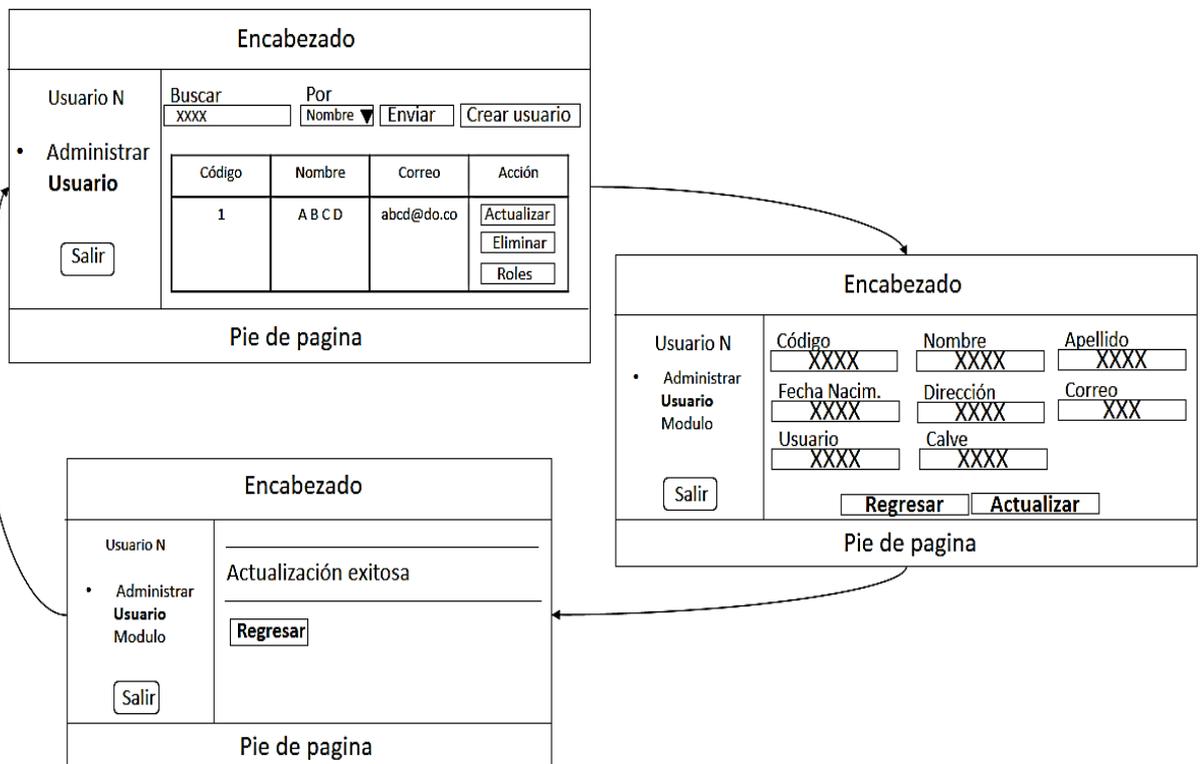


Figura 27. Storyboard actualizar usuario.  
Fuente: Autor del proyecto.

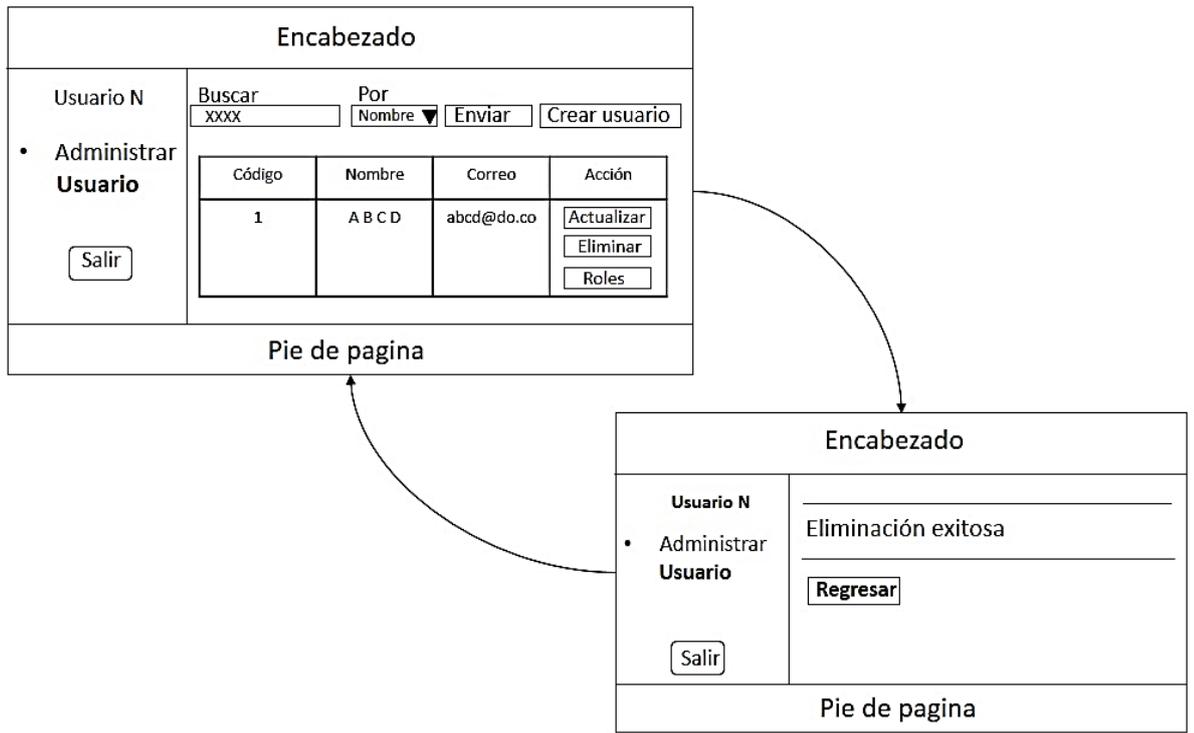


Figura 28. Storyboard eliminar usuario.  
Fuente: Autor del proyecto

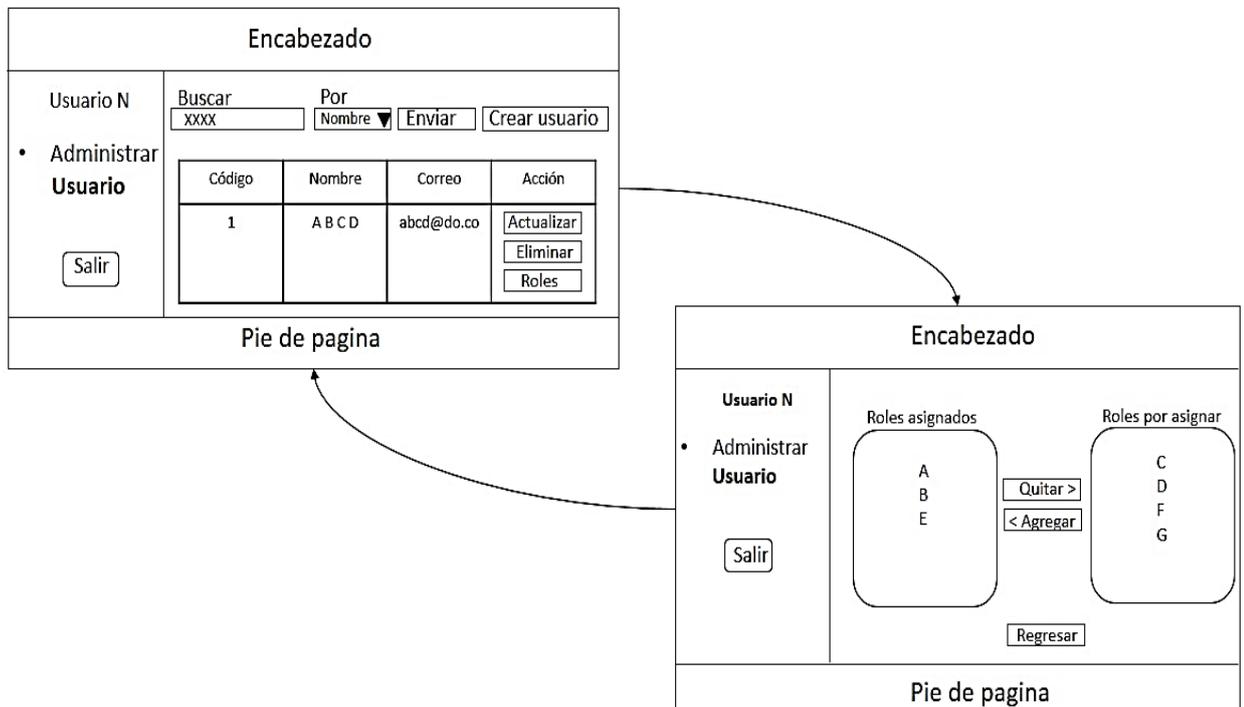


Figura 29. Storyboard asignar roles.  
Fuente: Autor del proyecto.

- Diagrama de secuencia entre frontend y el backend de la funcionalidad gestión de usuarios.

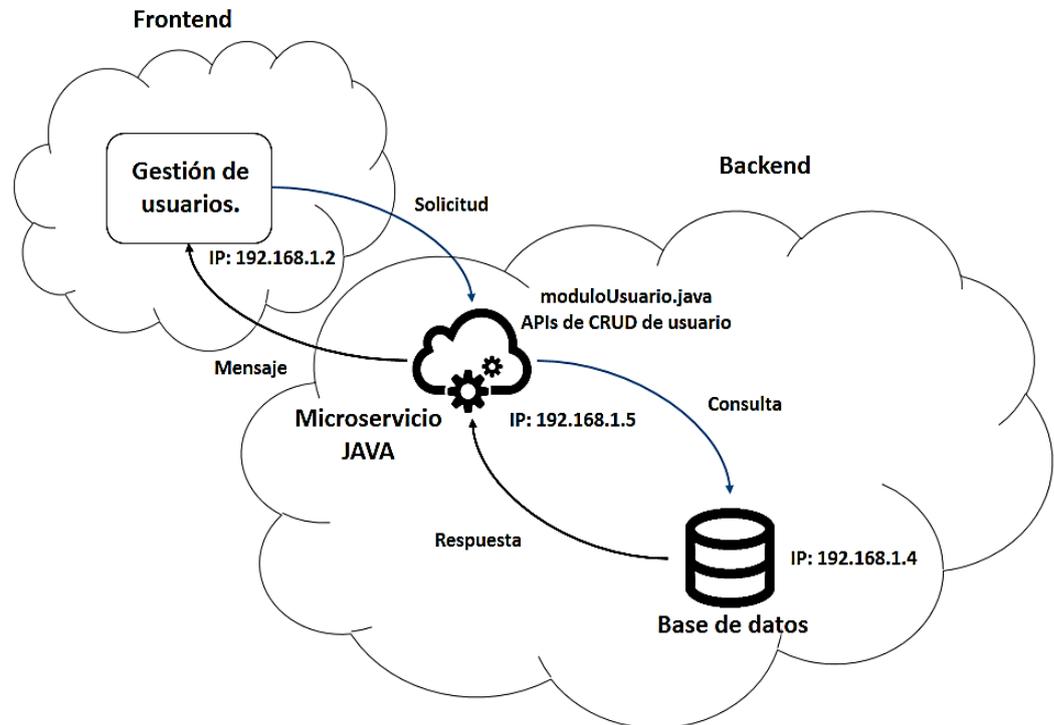


Figura 30. Diagrama de secuencia de la funcionalidad gestión de usuarios.  
Fuente: Autor del proyecto.

- Código implementado

Los métodos implementados a continuación, son los encargados de hacer las distintas operaciones **CRUD** en la base de datos, para ello se llevaron a cabo diferentes métodos tanto en el frontend como en el backend para esta funcionalidad.

- Método para buscar o listar usuarios desde el frontend.

```
$scope.buscarUsuario = function (dato) {
  $http({
    method: "GET",
    headers: {'Accept': 'application/json, text/plain',
      'Authorization': sesionControl.get("token")},
    url: APIURLJSP + "/api/usuario",
    params: {
      Criterio: dato.Criterio,
      Tipo: dato.Tipo,
      id: sesionControl.get("id")
    }
  }).then(function (response) {
    $scope.usuarios = response.data.datos;
    if ($scope.usuarios.length === 0) {
```

```

        alert("Sin informacion, intente de nuevo....");
    }
}, function (statusText) {
    if (statusText.status === 401) {
        sesionControl.clear();
        alert("Error: " + statusText.status + "\nMsg: "
            + statusText.data.error + "\n ==> No
            autorizado <==");
        window.location.replace('../');
    }
    $scope.usuarios = statusText;
});
};

```

Figura 31. Método para buscar usuario desde el frontend.  
Fuente: Autor del proyecto.

- o Métodos para consultar la solicitud del frontend mediante el backend en la base de datos.

```

@GET
@Produces("application/json")
public String listarUsuario(@QueryParam("Criterio") String
Criterio, @QueryParam("Tipo") int Tipo) throws JSONException {
    this.datos.put("Criterio", Criterio);
    this.datos.put("Tipo", Tipo);
    String msg = this.usuario.listarUsuario(this.datos);
    return msg;
}

```

Figura 32. Método para buscar usuario del API gestión de usuarios.  
Fuente: Autor del proyecto.

```

public String listarUsuario(HashMap datos) throws JSONException
{
    HashMap temp;
    StringBuilder sql = new StringBuilder();
    JSONObject jsn = new JSONObject();
    PreparedStatement ps;
    ResultSet rs;
    int index = 1;
    try {
        sql.append("select * from persona ");
        switch ((Integer) datos.get("Tipo")) {
            case 1:
                sql.append(" where per_id = ? ");
                break;
            case 2:
                sql.append(" where lower(per_nombre) like
                concat('%',lower(?),'%') ");
                break;
            case 3:
                sql.append(" where lower(per_apellido)
                like concat('%',lower(?),'%') ");
                break;
            case 4:
                sql.append(" where lower(per_correo) like
                concat('%',lower(?),'%') ");

```

```

        break;
    }
    sql.append("order by per_id asc");
    ps =
    conexion.getConnection().prepareStatement(sql.toString(
    ));
    ps.setString(index++, (String)
    datos.get("Criterio"));
    rs = ps.executeQuery();
    ArrayList<HashMap> lista = new ArrayList<HashMap>();
    if (rs.isBeforeFirst()) {
        while (rs.next()) {
            temp = new HashMap();
            temp.put("per_id",
            rs.getString("per_id"));
            temp.put("per_nombre",
            rs.getString("per_nombre"));
            temp.put("per_apellido",
            rs.getString("per_apellido"));
            temp.put("per_fecha_nacimiento",
            rs.getDate("per_fecha_nacimiento"));
            temp.put("per_direccion",
            rs.getString("per_direccion"));
            temp.put("per_correo",
            rs.getString("per_correo"));
            lista.add(temp);
        }
        jsn.put("datos", lista);
        rs.close();
        ps.close();
        conexion.cerrarConexion();
    } catch (SQLException e) {
        jsn.put("msg", e.getMessage());
    }
    return jsn.toString();
}

```

*Figura 33. Método de acceso a los datos (DAO).  
Fuente: Autor del proyecto.*

- Método para ingresar un nuevo usuario desde el frontend.

```

$scope.insertar = function (dato) {
    var date = $filter('date')(new
    Date(dato.per_fecha_nacimiento), 'yyyy-MM-dd');
    $http({
        method: "POST",
        headers: {'Accept': 'application/json, text/plain',
        'Authorization': sesionControl.get("token")},
        url: APIURLJSP + "/api/usuario",
        params: {
            per_id: dato.per_id,
            per_nombre: dato.per_nombre,
            per_apellido: dato.per_apellido,
            per_fecha_nacimiento: date,
            per_direccion: dato.per_direccion,
            per_correo: dato.per_correo,
            rol_descripcion: dato.rol_descripcion,

```

```

        usu_usuario: dato.usu_usuario,
        usu_clave: dato.usu_clave,
        id: sesionControl.get("id")
    }
}).then(function (response) {
    MyService.data.msgUsuario = response.data.msg;
    $state.go('msg');
}, function (statusText) {
    if (statusText.status === 401) {
        sesionControl.clear();
        alert("Error: " + statusText.status + "\nMsg: "
            + statusText.data.error + "\n ==> No
            autorizado <===");
        window.location.replace('../');
    }
    $scope.usuarios = statusText;
});
};

```

Figura 34. Método para registrar usuario desde el frontend.  
Fuente: Autor del proyecto.

- Métodos para realizar el registro del nuevo usuario en la base de datos.

```

@POST
public String registrar(
    @QueryParam("per_id") String per_id,
    @QueryParam("per_nombre") String per_nombre,
    @QueryParam("per_apellido") String per_apellido,
    @QueryParam("per_fecha_nacimiento") Date
    per_fecha_nacimiento,
    @QueryParam("per_direccion") String per_direccion,
    @QueryParam("per_correo") String per_correo,
    @QueryParam("usu_usuario") String usu_usuario,
    @QueryParam("usu_clave") String usu_clave
) throws JSONException {
    this.datos.put("per_id", per_id);
    this.datos.put("per_nombre", per_nombre);
    this.datos.put("per_apellido", per_apellido);
    this.datos.put("per_fecha_nacimiento",
        per_fecha_nacimiento);
    this.datos.put("per_direccion", per_direccion);
    this.datos.put("per_correo", per_correo);
    this.datos.put("usu_usuario", usu_usuario);
    this.datos.put("usu_clave", usu_clave);
    this.json.put("msg",
        this.usuario.registrarPersona(this.datos) +
        this.usuario.registrarUsuario(this.datos));
    String msg = this.json.toString();
    return msg;
}

```

Figura 35. Método para registrar usuario del API gestión de usuarios.  
Fuente: Autor del proyecto.

```

public String registrarPersona(HashMap datos) {
    String mensaje = "";
    StringBuilder sql = new StringBuilder();

```

```

try {
    PreparedStatement ps;
    int rs;
    int index = 1;
    sql.append("insert into persona "+
    "(per_id,per_nombre,per_apellido,per_fecha_nacimiento
    , per_direccion, per_correo ) "+ "values
    (?, ?, ?, ?, ?, ?) ");
    ps =
    this.conexion.getConnection().prepareStatement(sql.toString());
    ps.setString(index++, (String) datos.get("per_id"));
    ps.setString(index++, (String)
    datos.get("per_nombre"));
    ps.setString(index++, (String)
    datos.get("per_apellido"));
    ps.setDate(index++, (Date)
    datos.get("per_fecha_nacimiento"));
    ps.setString(index++, (String)
    datos.get("per_direccion"));
    ps.setString(index++, (String)
    datos.get("per_correo"));
    rs = ps.executeUpdate();
    if (rs > 0) {
        mensaje = "Registro ";
    }
    ps.close();
} catch (Exception e) {
    mensaje = e.getMessage();
}
return mensaje;
}

```

Figura 36. Método de acceso a los datos (DAO).  
Fuente: Autor del proyecto.

- o Método para actualizar un usuario desde el frontend.

```

$scope.actualizar = function (dato) {
    var date = $filter('date')(new
    Date(dato.per_fecha_nacimiento), 'yyyy-MM-dd');
    $http({
        method: "PUT",
        headers: {'Accept': 'application/json, text/plain',
        'Authorization': sesionControl.get("token")},
        url: APIURLJSP + "/api/usuario",
        params: {
            per_id: dato.per_id,
            per_nombre: dato.per_nombre,
            per_apellido: dato.per_apellido,
            per_fecha_nacimiento: date,
            per_direccion: dato.per_direccion,
            per_correo: dato.per_correo,
            rol_descripcion: dato.rol_descripcion,
            usu_usuario: dato.usu_usuario,
            usu_clave: dato.usu_clave,
            id: sesionControl.get("id")
        }
    })
}

```

```

    }).then(function (response) {
        MyService.data.msgUsuario = response.data.msg;
        $state.go("msg");
    }, function (statusText) {
        if (statusText.status === 401) {
            sesionControl.clear();
            alert("Error: " + statusText.status + "\nMsg: "
                + statusText.data.error + "\n ==> No
                autorizado <==");
            window.location.replace('../');
        }
        $scope.usuarios = statusText;
    });
};

```

Figura 37. Método para actualizar usuario desde el frontend.  
Fuente: Autor del proyecto.

- o Métodos para actualizar un usuario en la base de datos.

```

@PUT
public String actualizar(
    @QueryParam("per_id") String per_id,
    @QueryParam("per_nombre") String per_nombre,
    @QueryParam("per_apellido") String per_apellido,
    @QueryParam("per_fecha_nacimiento") Date
    per_fecha_nacimiento,
    @QueryParam("per_direccion") String per_direccion,
    @QueryParam("per_correo") String per_correo,
    @QueryParam("usu_usuario") String usu_usuario,
    @QueryParam("usu_clave") String usu_clave
) throws JSONException {
    this.datos.put("per_id", per_id);
    this.datos.put("per_nombre", per_nombre);
    this.datos.put("per_apellido", per_apellido);
    this.datos.put("per_fecha_nacimiento",
        per_fecha_nacimiento);
    this.datos.put("per_direccion", per_direccion);
    this.datos.put("per_correo", per_correo);
    this.datos.put("usu_usuario", usu_usuario);
    this.datos.put("usu_clave", usu_clave);
    this.json.put("msg",
        this.usuario.actualizarPersona(this.datos) +
        this.usuario.actualizarUsuario(this.datos));
    String msg = this.json.toString();
    return msg;
}

```

Figura 38. Método para actualizar usuario del API gestión de usuarios.  
Fuente: Autor del proyecto.

```

public String actualizarPersona(HashMap datos) {
    String mensaje = "";
    StringBuilder sql = new StringBuilder();
    try {
        PreparedStatement ps;
        int rs;
        int index = 1;
    }
}

```

```

sql.append("update persona " + "set per_nombre=?,
per_apellido=?, per_fecha_nacimiento=?,
per_direccion=?, per_correo=? "+ "where per_id=?");
ps =
this.conexion.getConnection().prepareStatement(sql.toString());
ps.setString(index++, (String)
datos.get("per_nombre"));
ps.setString(index++, (String)
datos.get("per_apellido"));
ps.setDate(index++, (Date)
datos.get("per_fecha_nacimiento"));
ps.setString(index++, (String)
datos.get("per_direccion"));
ps.setString(index++, (String)
datos.get("per_correo"));
ps.setString(index++, (String) datos.get("per_id"));
rs = ps.executeUpdate();
if (rs > 0) {
    mensaje = "Actualizacion ";
}
ps.close();
} catch (Exception e) {
    mensaje = e.getMessage();
}
return mensaje;
}

```

Figura 39. Método de acceso a los datos (DAO).  
Fuente: Autor del proyecto.

- Método para eliminar un usuario desde el frontend.

```

$scope.eliminar = function (dato) {
    $http({
        method: "DELETE",
        headers: {'Accept': 'application/json, text/plain',
        'Authorization': sesionControl.get("token")},
        url: APIURLJSP + "/api/usuario",
        params: {per_id: dato, id: sesionControl.get("id")}
    }).then(function (response) {
        MyService.data.msgUsuario = response.data.msg;
        $state.go("msg");
    }, function (statusText) {
        if (statusText.status === 401) {
            sesionControl.clear();
            alert("Error: " + statusText.status + "\nMsg: "
            + statusText.data.error + "\n ===> No
            autorizado <===");
            window.location.replace('../');
        }
    });
    $scope.usuarios = statusText;
};

```

Figura 40. Método para eliminar usuario desde el frontend.  
Fuente: Autor del proyecto.

- Métodos para eliminar un usuario en la base de datos.

```
@DELETE
public String eliminar(@QueryParam("per_id") String per_id)
throws JSONException {
    this.datos.put("per_id", per_id);
    this.json.put("msg",
    this.usuario.eliminarPersona(this.datos));
    String msg = this.json.toString();
    return msg;
}
```

Figura 41. Método para eliminar usuario del API gestión de usuarios.  
Fuente: Autor del proyecto.

```
public String eliminarPersona(HashMap datos) {
    String mensaje = "";
    StringBuilder sql = new StringBuilder();
    try {
        PreparedStatement ps;
        int rs;
        int index = 1;

        sql.append("delete from persona where per_id=?");
        ps =
        this.conexion.getConexion().prepareStatement(sql.toString());
        ps.setString(index++, (String) datos.get("per_id"));
        rs = ps.executeUpdate();
        if (rs > 0) {
            mensaje = "Eliminacion con exito";
        }
        ps.close();
    } catch (Exception e) {
        mensaje = e.getMessage();
    }
    return mensaje;
}
```

Figura 42. Método de acceso a los datos (DAO).  
Fuente: Autor del proyecto.

#### 4.4 Análisis de validación de la estrategia

Desde el punto de vista del desarrollo, se puede determinar que la implantación del prototipo ha permitido validar satisfactoriamente la estrategia, ya que este fue desarrollado en base a su planteamiento, pues su desempeño fue óptimo gracias a la ejecución de los diferentes microservicios que permitieron integrar varias tecnologías web, viéndose como una sola.

De igual forma, permite a los programadores tener un desarrollo más ordenado del trabajo para implementar y realizar cambios en el software de forma rápida y sencilla en los numerosos módulos que contiene la aplicación web, pues cada

módulo es independiente de los demás y ante cualquier fallo, este se puede detectar de forma rápida sin tener que analizar uno por uno.

En este sentido, se puede concluir que su implementación permite dividir el trabajo en grupos de programadores, pues cada grupo puede ser seleccionado para desarrollar servicios independientes (microservicios) sin afectar los otros, ya que cada servicio puede ser desarrollado bajo cualquier tecnología web y su soporte no afectaría a los otros.

Esto indica que la estrategia permite el desarrollo de aplicaciones web rápidamente, ya que cada microservicio podrá ser construido bajo la tecnología más óptima para su desempeño.

Finalmente, la estructura de su implantación puede ser entendible y modificable por terceros de forma fácil, pues cada funcionalidad puede ser sencilla de administrar.

## 5 Conclusiones

Se pudo determinar que la revisión bibliográfica por las diferentes tecnologías web existentes tanto del frontend como backend, fue de gran importancia para este trabajo, ya que sus características permitieron seleccionar las herramientas apropiadas para la construcción del prototipo que se elaboró en base a la estrategia y tuvo un buen desempeño al integrar las diferentes tecnologías en una sola aplicación web.

Es de gran importancia conocer los diferentes modelos que proponen los autores para integrar tecnologías web, estos permitieron tener una clara idea de su funcionamiento y comportamiento. De este modo, se definió una nueva estrategia de integración de plataformas web dinámicas con el fin de construir aplicaciones que incluyan más de una tecnología con diversas funcionalidades para que los usuarios realicen diferentes operaciones en la aplicación web mejor rendimiento en su navegación, sin que se den cuenta a que servicio están accediendo.

La validación de la estrategia se realizó con el desarrollo de un prototipo web, el cual se basó en la distribución de funcionalidades en diferentes máquinas, desarrolladas con la implementación de microservicios bajo la arquitectura REST para el backend y AngularJS del lado del frontend, las cuales tuvieron un comportamiento aceptable por lo propuesto en la estrategia. Esto ayudará a tener un control más eficiente al momento de realizar futuras modificaciones en las funcionalidades que se llevaron a cabo para construir aplicaciones web sin afectar las demás cuando se integre un nuevo recurso a la aplicación ya que estos trabajan con ayuda de microservicios independientes.

## 6 Recomendaciones y Trabajos Futuros

Se recomienda para trabajos futuros hacer un análisis de los diferentes frameworks existentes de desarrollo para ahorrar tiempo y estructuración de código en la construcción de microservicios en el backend, ya que estos no se analizaron en el transcurso de este trabajo.

De igual manera, se puede analizar en la estrategia que esta no incluye una seguridad completa para proteger los datos ante posibles ataques que se hagan a los servidores. En base a esto, es necesario que se lleve a cabo una nueva propuesta para brindarle seguridad a la estrategia que se planteó en este documento con el fin de minimizar en un 99 por ciento de los fallos que presentan hoy en día en las aplicaciones web

Igualmente sería de gran aporte para la comunidad estudiantil y docente del programa de ingeniería de sistemas, hacer uso de esta estrategia para conocer las diferentes tecnologías web que fueron implementadas en este proyecto y crear aplicaciones web orientadas a microservicios para dividir el frontend del backend de los proyectos web monolíticos que se usan hoy en día.

Actualmente el desarrollo web ha crecido de forma rápida, pues las nuevas tecnologías han facilitado la construcción de aplicaciones web de acuerdo a las necesidades del cliente, una de estas tecnologías a destacar se basa en migrar contenido de páginas web a aplicaciones móviles donde solo se va utilizar lenguajes de programación web ya sea HTML, CSS y JavaScript , ya que estas permiten un acceso rápido a la información, para ello se recomienda hacer un estudio de las diferentes tecnologías existentes para la maquetación de App híbridas basadas en Apache Cordova, Phonegap o Ionic Framework (basado en AngularJS), que puedan hacer uso de la estrategias propuesta en este trabajo y determinar su comportamiento cuando se implementen en distintos sistemas operativos móviles tales como Android, iOS o Windows Phone.

## 7 Bibliografía

Chapaval, Nicole. “Qué es Frontend y Backend” [En línea], Febrero de 2018 [revisado 8 de junio de 2018]. Disponible en: <https://platzi.com/blog/que-es-frontend-y-backend/>

Lázaro, Pablo. “¿Qué es AngularJS? Una breve introducción” [En línea], Mayo de 2013 [revisado 8 de junio de 2018]. Disponible en: <http://pablolazarodev.blogspot.com/2013/05/que-es-angularjs-una-breve-introduccion.html>

Grados Caballero, Julio Giampiere. “¿Cómo funciona React.js?” [En línea], [revisado 8 de junio de 2108]. Disponible en: <https://devcode.la/blog/como-funciona-reactjs/>

Macías Rodríguez, Rafael. “Primeros pasos para conocer Emberjs” [En línea], Abril de 2013 [revisado 9 de junio de 2018]. Disponible en: <https://www.adictosaltrabajo.com/tutoriales/1-tutorial-emberjs/>

You, Evan. “VUE.JS” [En línea], [revisado 9 de junio de 2018]. Disponible en: <https://vuejs.org/v2/guide/>

Rosas, Jesús. “Que es Backbone.js y como funciona” [En línea], Noviembre de 2018 [revisado 10 de junio de 2018]. Disponible en: <https://www.azulweb.net/que-es-backbone-js-y-como-funciona/>

Bulkar, Mangesh. “The Difference Between Static and Dynamic Websites?” [En línea], Diciembre de 2015 [revisado 10 de junio de 2018]. Disponible en: <https://www.linkedin.com/pulse/difference-between-static-dynamic-websites-mangesh-bulkar>

Rouse, Margaret. “HTTP (Hypertext Transfer Protocol)” [En línea], [revisado 10 de junio de 2018]. Disponible en: <https://searchwindevelopment.techtarget.com/definition/HTTP>

Zaforas, Manuel. “¿Es Python el lenguaje del futuro?” [En línea], Noviembre de 2017 [revisado 11 de junio de 2018]. Disponible en: <https://www.paradigmadigital.com/dev/es-python-el-lenguaje-del-futuro/>

The PHP Group. “¿Qué es PHP?” [En línea], 2018 [revisado 11 de junio de 2018]. Disponible en: <http://php.net/manual/es/intro-what-is.php>

Beal, Vangie. “Java” [En línea], [revisado 11 de junio de 2018]. Disponible en: <https://www.webopedia.com/TERM/J/Java.html>

Rouse, Margaret. “Ruby” [En línea], [revisado 11 de junio de 2018]. Disponible en: <https://what-is.techtarget.com/definition/Ruby>

Microsoft. “Introducción al lenguaje C# y .NET Framework” [En línea], Julio de 2015 [revisado 12 de junio de 2018]. Disponible en: <https://docs.microsoft.com/es-es/dotnet/csharp/getting-started/introduction-to-the-csharp-language-and-the-net-framework>

JSON. “Introducing JSON” [En línea], [revisado 12 de junio de 2018]. Disponible en: <https://www.json.org/>

W3.org. “Extensible Markup Language (XML)” [En línea], [revisado el 12 de junio de 2018]. <https://www.w3.org/XML/>

Gavin, Brady. “What Is An XML File (And How Do I Open One)?” [En línea], [revisado 12 de junio de 2018]. Disponible en: <https://www.howtogeek.com/357092/what-is-an-xml-file-and-how-do-i-open-one/>

Lewis, James y Fowler, Martin. “Microservices a definition of this new architectural term” [En línea], Marzo de 2014 [revisado 13 de junio de 2018]. Disponible en: <https://martinfowler.com/articles/microservices.html>

Álvarez Caules, Cecilio. “¿Qué es un Microservicio?” [En línea], Enero de 2015 [revisado 13 de junio de 2018]. Disponible en: <https://www.arquitecturajava.com/que-es-un-microservicio/>

Juanda. “SPA” [En línea], [revisado 13 de junio de 2018]. Disponible en: [https://juanda.gitbooks.io/webapps/content/spa/arquitectura\\_de\\_un\\_spa.html](https://juanda.gitbooks.io/webapps/content/spa/arquitectura_de_un_spa.html)

Navarro Maset, Rafael. “Modelado, Diseño e Implementación de Servicios Web” [En línea], Julio de 2007 [revisado 14 de junio de 2018]. Disponible en: <http://users.dsic.upv.es/~rnavarro/NewWeb/docs/RestVsWebServices.pdf>

Raymundo, José – Rodríguez, Abelardo – Andrade, Héctor – Rivera, Rafael. “Modelo de integración de tecnologías web para la gestión de contenido virtual B2B” [En línea], Enero de 2013 [revisado 14 de junio de 2018]. Disponible en: [https://www.researchgate.net/publication/322298299\\_Modelo\\_de\\_integracion\\_de\\_tecnologias\\_web\\_para\\_la\\_gestion\\_de\\_contenido\\_virtual\\_B2B](https://www.researchgate.net/publication/322298299_Modelo_de_integracion_de_tecnologias_web_para_la_gestion_de_contenido_virtual_B2B)

Pérez-Herrera Cuadrillero, Manuel. “Arquitecturas basadas en microservicios” [En línea], Agosto de 2015 [revisada 15 de junio de 2018]. Disponible en: <http://oa.upm.es/37346/>

Valdivia Caballero, José Jesús. “Modelo de procesos para el desarrollo del front-end de aplicaciones web” [En línea], Octubre de 2016 [revisado 15 de junio de 2018]. Disponible en: <https://dialnet.unirioja.es/descarga/articulo/6043088.pdf>

Lopera Martínez, Ana Isabel. “Diseño y desarrollo del backend para la explotación de componentes web empleando Google App Engine” [En línea],

Junio de 2015 [revisado 16 de junio de 2018]. Disponible en: <http://oa.upm.es/38349/>

Birch, Thomas S. "Proceso software de una aplicación web de apoyo a la oración" [En línea], Julio de 2017 [revisado 17 de junio de 2018]. Disponible en: <http://oa.upm.es/49167/>

Moreno Fernández-Cañadas, José María. "Aplicación Android de recetas de cocina con backend Symfony" [En línea], Julio de 2017 [revisado 17 de junio de 2018]. Disponible en: <http://oa.upm.es/48305/>

López, Daniel – Maya, Edgar. "Arquitectura de Software basada en Microservicios para Desarrollo de Aplicaciones Web" [En línea], [revisado 17 de junio de 2018]. Disponible en: <https://documentos.redclara.net/bitstream/10786/1277/1/>

Emba Hernández, José Manuel. "Sistema de filtrado web inteligente basado en un sistema multi-agente" En línea], Noviembre de 2014 [revisado 17 de junio de 2018]. Disponible en: <https://tesis.ipn.mx/jspui/handle/123456789/22650?mode=full>

Ortega Muñoz, Julián Camilo. "CLUBMAT 1.1: Extensión de una aplicación web destinada al fortalecimiento de clubes escolares matemáticos integrando JavaFX2 y JavaEE6 con servicios web basados en REST" [En línea], Octubre de 2013 [revisado 18 de junio de 2018]. Disponible en: <http://pegasus.javeriana.edu.co/~CIS1330IS04/documentos/MemoriaTrabajoGrado-JulianOrtega.pdf>

Nazmutdinov, Stanislav. "Web application front end architecture and development using angularjs framework" [En línea], 2015 [revisado 18 de junio de 2018]. Disponible en: <https://digi.lib.ttu.ee/i/file.php?DLID=3453&t=1>