- CAPÍTULO 2

- MARCO TEÓRICO

- 2.1 Modelo de negocio

- 2.1.1 Modelo de negocio actual

- 2.1.2 Modelo de negocio alternativo

- 2.1.3 Diseño del modelo de negocio

- 2.2 Ingeniería de Software

- 2.2.1 Metodologías de desarrollo de software

- 2.2.2 Metodologías de desarrollo ágil

- a) Programación Extrema (XP)

- b) SCRUM

- 2.2.3 Selección de la metodología de desarrollo

- 2.3 Arquitectura de software

- 2.3.1 Comparaciones

- 2.3.2.1 Front-end y back-end

- 2.3.2.2 Aplicación de una sola página (SPA)

- 2.3.2.3 Back-end para Front-end (BFF)

- 2.3.2 Arquitectura orientada a Servicios (SOA)

- 2.3.3 Arquitectura multi/N capas (MLA)

- 2.3.4 Arquitectura de Microservicios (MSA)

- 2.3.5 Justificación de la arquitectura

- 2.4 Aplicaciones web

- 2.4.1 Arquitectura Cliente/Servidor

- 2.4.2 Servicios web

- 2.4.2.1 Justificación del servicio web e intercambio de datos

- 2.4.2.2 SOAP (Simple Objet Access Protocol)

- 2.4.2.3 REST (Representational State Transfer)

- 2.4.3 Intercambio de datos

- 2.4.3.1 Lenguaje de marcado extensible (XML)

- 2.4.3.2 Notación de objetos de JavaScript (JSON)

- 2.5 Tipos de aplicaciones móviles

- 2.5.1 Aplicaciones nativas

- 2.5.2 Aplicaciones web móviles

- 2.5.3 Aplicaciones híbridas

- 2.5.4 Elección de la aplicación móvil

- 2.6 Indexación unica

- 2.6.1 Identificador único universal UUID

- 2.6.2 Códigos de respuesta rápida QR

- 2.7 Lenguajes de programación

- 2.7.1 C Sharp (C#)

- 2.7.2 Java

- 2.7.3 JavaScript

- 2.7.4 Justificacion de los lenguajes de programación

- 2.7.5 Conclusión

- 2.7.6 Elección del lenguaje de programación

- 2.8 Calidad del software

- 2.8.1 Pruebas unitarias

- 2.8.2 Usabilidad

- 2.8.3 Seguridad del sistema

**CAPÍTULO 2**

**MARCO TEÓRICO**

En este capítulo se analizará las diferente arquitecturas de desarrollo de software que se utilizan actualmente, velando de los principios del desarrollo en calidad, escalabilidad seguridad y mantenimiento, del software y la curva de aprendizaje del entorno de desarrollo para los programadores, así también los patrones, anti-patrones de diseño, que surgieron a largo de los últimos años.

**2.1 Modelo de negocio**

El "Modelo de Negocio" se define como un proceso de representación de uno o más aspectos o elementos de una empresa u organización. Se busca modelar las actividades que se hacen dentro de una empresa la cual va a tener un fin; los aspectos que se desean modelar son su propósito, estructura, funcionalidad, dinámica, logística de negocios y sus componentes como fines, procesos de negocio, regla de negocio, objeto de negocio, actores y unidades organizativas [1].

En el desarrollo de software, los requisitos tienen lugar en el espacio de la solución; el Modelado de Negocios aporta información esencial para la ingeniería de requisitos [2]. Siendo así la caracterización de los aspectos más significativos de la empresa o una parte de ella, para ello se debe tener claro cuál es el fin que se busca con ese modelo, para así tener claro los elementos del negocio que se desean representar y puede ser orientado al negocio propiamente u orientadas a la tecnología y su aplicación en el negocio.

**2.1.1 Modelo de negocio actual**

El modelo de negocio actual es una parte primordial de cualquier proceso de desarrollo de software, permite lograr comprender el ámbito de la información e identificar las responsabilidades, actividades y procesos que se realizan dentro de la organización. Éste resultado es de gran ayuda para identificar las posibles formas de hacer el negocio más eficiente.

**2.1.2 Modelo de negocio alternativo**

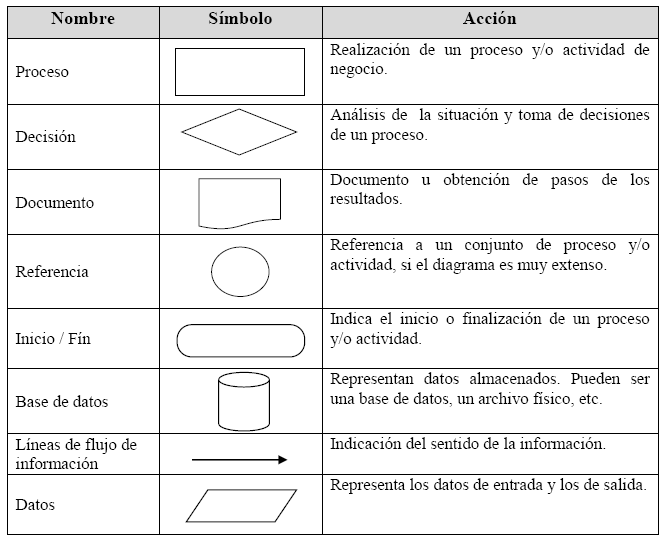
El modelo de negocio alternativo es la representación de la situación mejorada, lo cual requiere el análisis de los procesos del modelo de negocio actual y la búsqueda del modelo para nuevas formas de hacer las cosas. Esto significa que modelo de negocio actual y sus procesos son cambiados significativamente para crear diferentes y mejorados procesos.

**2.1.3 Diseño del modelo de negocio**

Los diagramas de flujo permiten modelar y representar gráficamente todo tipo de sistemas, una de las características principales es que proporcionan información sobre los procesos de forma clara, ordenada y concisa.

**Cuadro 2.1**

**Elementos principales de un diagrama de flujo**

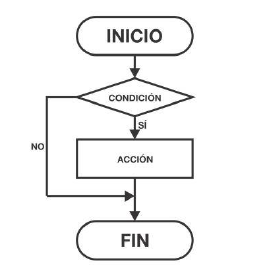
[](https://github.com/wolf-mtwo/lagash-documentation/blob/dev/anexos/cuadro-1.png)

***Fuente:****[Elaboración propia]*

A continuación se muestra un modelo de básico de un diagrama de flujo:

**Figura 2.1**

**Diagrama de flujo**

[](https://github.com/wolf-mtwo/lagash-documentation/blob/dev/anexos/figura-1.png)

***Fuente:****[Elaboración propia]*

**2.2 Ingeniería de Software**

La ingeniería de software es "una disciplina que comprende todos los aspectos de la producción de software desde las etapas iníciales de la especificación del sistema, hasta el mantenimiento de este después de que se utiliza" [3]. El IEEE16 (Instituto de Ingeniería Eléctrica y Electrónica) define que: "la ingeniería de software es la aplicación de un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación y mantenimiento de software [4]. Según Pressman la ingeniería de software es una tecnología con varias capas.

**Figura 2.2**

**Capas de la ingeniería de software**

[](https://github.com/wolf-mtwo/lagash-documentation/blob/dev/anexos/figura-2.png)

***Fuente:****[11, Fig. 1.3]*

Cualquier enfoque de ingeniería (incluso la de software) debe basarse en un compromiso organizacional con la calidad [4].

La ingeniería de software es de mucha importancia ya que sugiere que se utilicen ciertas estrategias como metodologías, métodos, técnicas y tecnologías de software, dando una esencia del verdadero proceso de desarrollo de software las cuales serán aplicadas para la resolución de problemas del proyecto describiendo algunos modelos de desarrollo y sus procesos.

**2.2.1 Metodologías de desarrollo de software**

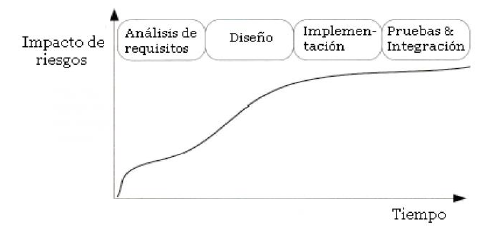
La metodología de software es un conjunto de procedimientos, técnicas, herramientas y un soporte documental que ayuda a los desarrolladores a realizar nuevo software. La elección de metodología o procesos de desarrollo de software es de trascendental importancia, porque es la que nos guiara para alcanzar el objetivo de culminar el proyecto. Para eso se planteara una comparación entre las diversas metodologías. Para el presente proyecto desde inicio se consideró aplicar algún proceso de desarrollo ágil principalmente debido a un alto grado de incertidumbre en cuanto a los requerimientos funcionales se refiere. Sin embargo, es necesario realizar un breve análisis de modelos de desarrollo ágil existentes y tomar la decisión correcta respecto al proceso que más se adecua a las condiciones del proyecto. Por lo tanto, a continuación se describen tres modelos candidatos a ser aplicados en el proyecto y posteriormente se comparan sus características principales.

**2.2.2 Metodologías de desarrollo ágil**

El desarrollo ágil de software es un grupo de metodologías de desarrollo de software que se basan en principios similares. Las metodologías ágiles promueven generalmente un proceso de gestión de proyectos que fomenta el trabajo en equipo, la organización y responsabilidad propia, un conjunto de mejores prácticas de ingeniería que permiten la entrega rápida de software de alta calidad, y un enfoque de negocio que alinea el desarrollo con las necesidades del cliente. Potencialmente, casi todos los métodos ágiles son adecuados para la adaptación. La conveniencia de situación puede considerarse como una característica de distinción entre los métodos ágiles y los métodos tradicionales, que son mucho más rígidos y preceptivos. Asimismo, las metodologías convencionales tienden a acumular los riesgos y dificultades que surgen en el desarrollo del producto al final del proyecto, como puede apreciarse en la **figura 2.3**, repercutiendo en retrasos en la entrega de productos o influyendo en la incorrecta ejecución de las últimas fases del ciclo de vida.

**Figura 2.3**

**Distribución del riesgo**

[](https://github.com/wolf-mtwo/lagash-documentation/blob/dev/anexos/figura-3.png)

***Fuente:****[5]*

Se puede decir que las metodologías tradicionales ponen por encima de lo demás los objetivos de la definición y del control del trabajo, mientras que las metodologías ágiles priman la libertad del equipo, la comunicación entre el cliente y el equipo, realizar sólo las tareas que aportan valor al cliente, y finalmente definir el trabajo tal y como éste se va realizando para el conocimiento adquirido durante su desarrollo evite realizar tareas innecesarias. A continuación se presentaran diferentes modelos agiles de proceso. Entre ellas existen mucha similitud sin embargo cada una posee características únicas que las diferencian de las demás.

**a) Programación Extrema (XP)**

La programación extrema, es una metodología de desarrollo ágil, una de las más exitosas en tiempo reciente. La programación extrema (XP) es un enfoque de la ingeniería del software formulado por Kent Beck. Es el más destacado de los procesos ágiles de desarrollo de software. Se diferencia de las metodologías tradicionales principalmente en que pone más énfasis en la adaptabilidad que en la previsibilidad.

El objetivo principal de XP es la satisfacción del cliente. Se le trata de dar al cliente lo que quiere y cuando quiere. Por tanto, se debe responder rápidamente a las necesidades del cliente. Como metodología Ágil que es, se pueden producir modificaciones de los requisitos del proyecto a lo largo de su desarrollo, sin que esto produzca dificultades. Pressman dice que "La programación extrema usa un enfoque orientado a objeto como paradigma preferido de desarrollo, y engloba un conjunto de reglas y prácticas que ocurren en el contexto de cuatro actividades estructurales: planeación, diseño, codificación y pruebas" [6, pp. 61].

XP es una metodología centrada en potenciar las relaciones interpersonales como clave para el éxito en el desarrollo de software, promoviendo el trabajo en equipo y basándose en la realimentación continua entre el cliente y el desarrollador. XP es adecuado para proyectos con requisitos imprecisos y cambiantes.

**b) SCRUM**

SCRUM es una metodología que nace ajena al desarrollo del software, de hecho sus principios fundamentales fueron desarrollados en procesos de reingeniería por Goldratt, Takeuchi y Nonaka en la década de 1980. Scrum es un proceso ágil que se puede usar para gestionar controlar desarrollos complejos de software y productos usando prácticas iterativas e incrementales. Es un proceso incremental iterativo para desarrollar cualquier producto o gestionar cualquier trabajo. Los actores contemplados en esta metodología son los siguientes:

* **Product Owner**, conoce y marca las prioridades del proyecto o producto.
* **Scrum Master**, es la persona que asegura el seguimiento de la metodología guiando las reuniones y ayudando al equipo ante cualquier problema que pueda aparecer. Su responsabilidad es entre otras, la de hacer de paraguas ante las presiones externas.
* **Scrum Team**, son las personas responsables de implementar la funcionalidad o funcionalidades elegidas por el Product Owner.
* **Usuarios o Cliente**, son los beneficiarios finales del producto, y son quienes viendo los progresos, pueden aportar ideas, sugerencias o necesidades. que incluye a los desarrolladores.

**2.2.3 Selección de la metodología de desarrollo**

Entre las 2 metodologías ágiles descritas anteriormente la más utilizada es la metodología XP, ya que esta metodología propone una estrecha relación entre el equipo de desarrollo y los clientes, estableciendo como base la comunicación.

En el presente proyecto se aplicará la metodología ágil XP, cuya característica principal consiste en contemplar y dar respuesta a las necesidades dinámicas del cliente. Mediante la identificación y reducción del riesgo por medio de un desarrollo iterativo, con capacidad de respuesta ante cambios, permitiendo la adaptación a nuevos requisitos de la organización, mostrando resultados tangibles, logrando de esta forma efectividad y reduciendo riesgos, pero sin eliminar, la necesidad de documentación escrita.

**2.3 Arquitectura de software**

Arquitectura de software es el proceso de definir una solución estructural, que conoce todas las técnicas y requerimientos operacionales, optimizando atributos comunes de cualidad como performance, seguridad y manejabilidad. esto envuelve una serie de decisiones basadas en un amplio rango de factores y cada uno de estas decisiones puede tener considerable impacto, sobre todo en el éxito de la aplicación.

"Arquitectura de software abarca el conjunto de decisiones importantes acerca de la organización de un sistema de software que incluye la selección de los elementos estructurales y sus interfaces mediante el cual el sistema se compone; comportamiento como se especifica en la colaboración entre dichos elementos; composición de estos elementos estructurales y de comportamiento en subsistemas más grandes; y un estilo arquitectónico que guía a esta organización. arquitectura de software también incluye la funcionalidad, facilidad de uso, la resistencia, el rendimiento, la reutilización, la comprensibilidad, las limitaciones económicas y tecnológicas, ventajas y desventajas y las preocupaciones estéticas". [13]

Fowlwer, uno de los ingenieros más reconocidos en el área de desarrollo de software, dice que la arquitectura es "el desglose de más alto nivel de un sistema en sus partes; las decisiones que son difíciles de cambiar; hay múltiples arquitecturas en un sistema; lo que es de gran importancia arquitectónica puede cambiar durante la vida de un sistema; y, al final, la arquitectura se reduce a lo que más importante del sistema sea" [8].

**2.3.1 Comparaciones**

La creación de un sitio web o una página de internet en general implica dos factores, el *front-end* y el *back-end*.

*Front-end* y *back-end* son términos utilizados para caracterizar las interfaces del programa y los servicios relativos al usuario inicial de estas interfaces y servicios. (El "usuario" puede ser un ser humano o un programa.) Una aplicación *front-end* es aquella con la que los usuarios de la aplicación interactúan directamente.

**2.3.2.1 Front-end y back-end**

El *front-end* es todo lo relacionado con lo que el usuario ve, incluyendo el diseño y algunos lenguajes como HTML y CSS. Cuando discutimos el *front-end* de la web, lo que realmente estamos hablando es la parte de la web con la que puedes ver e interactuar. El *front-end* generalmente consta de dos partes: el diseño web y el desarrollo web front-end. En el pasado, cuando alguien hablaba de desarrollo por lo general se refiere al backend, pero en los últimos años ha habido una necesidad real de diferenciar entre los diseñadores que trabajaban estrictamente en Photoshop y los que podían codificar HTML y CSS. Fue aún más lejos cuando los diseñadores cruzaron las líneas para trabajar con JavaScript y jQuery.

El *back-end*, o el "lado del servidor", es básicamente cómo funciona el sitio, las actualizaciones. Esto se refiere a todo lo que el usuario no puede ver en el navegador, como bases de datos y servidores. En el lado de back-end los desarrolladores se preocupan más por cuestiones como la seguridad, la estructura y la gestión de contenidos.

Una aplicación o programa *back-end* sirve indirectamente en apoyo de los servicios front-end, normalmente por estar más cerca del recurso requerido o tener la capacidad de comunicarse con el recurso requerido. La aplicación de del lado del servidor puede interactuar directamente con el *front-end*.

El *back-end* normalmente consta de tres partes: un servidor, una aplicación y una base de datos. Si reserva un vuelo o compra entradas de conciertos, normalmente abre un sitio web e interactúa con el *front-end*. Una vez que haya ingresado esa información, la aplicación la almacena en una base de datos que se creó en un servidor.

Toda esa información permanece en el servidor, así que cuando vuelva a iniciar sesión en la aplicación para imprimir sus tickets, toda la información aún está en su cuenta.

**2.3.2.2 Aplicación de una sola página (SPA)**

La aplicación de una sola página son aplicaciones Web que cargan una sola página HTML y actualizan dinámicamente esa página a medida que el usuario interactúa con la aplicación. Las SPA utilizan AJAX y HTML5 para crear aplicaciones Web fluidas y sensibles, sin recargas constantes de la página. Sin embargo, esto significa que gran parte del trabajo ocurre en el lado del cliente, Afortunadamente, hay muchos frameworks en JavaScript de código libre que facilitan la creación de SPAs. Entre los más conocidos están Angular, Angular 2 y Ember.

**2.3.2.3 Back-end para Front-end (BFF)**

Los servicios soportan muchas variaciones en los clientes, como el móvil frente a la web y diferentes formas de interfaz web. Es tentador diseñar una única API de *back-end* para apoyar a todos los clientes con una API reutilizable. Pero las necesidades del cliente varían, al igual que las restricciones, como el ancho de banda para los dispositivos móviles frente al deseo de un montón de datos sobre las conexiones web rápidas. Por lo tanto, a menudo es mejor definir diferentes servicios de back-end para cada tipo de cliente front-end. Estos extremos posteriores deben ser desarrollados por equipos alineados con cada extremo frontal para asegurarse de que cada extremo posterior cumple adecuadamente las necesidades de su cliente.

**2.3.2 Arquitectura orientada a Servicios (SOA)**

La arquitectura orientada a servicios es un enfoque utilizado para crear una arquitectura basada en el uso de servicios. Los servicios (como los servicios Web RESTful) realizan alguna función pequeña, como la producción de datos, la validación de un cliente o la prestación de servicios analíticos simples.

Además de construir y exponer servicios, SOA tiene la capacidad de aprovechar estos servicios una y otra vez dentro de las aplicaciones (conocidas como aplicaciones compuestas). SOA abordando la mayoría de los principales sistemas como servicios, y la abstracción de los servicios en un único dominio donde se forman en soluciones.

Las arquitecturas orientadas al servicio no son algo nuevo. La primera arquitectura orientada a servicios para muchas personas en el pasado fue con el uso de DCOM u ORB (Object Request Brokers) basado en la especificación CORBA.

**2.3.3 Arquitectura multi/N capas (MLA)**

La arquitectura de una aplicación N-capas se caracteriza por la descomposición funcional de las aplicaciones, los componentes de servicio y su implementación distribuida, proporcionando escalabilidad, disponibilidad, capacidad de administración y utilización de recursos mejoradas. Cada nivel es completamente independiente de todos los demás niveles, excepto los inmediatamente superiores e inferiores. La comunicación entre niveles es típicamente asíncrona para soportar mejor escalabilidad. Las arquitecturas de nivel N generalmente tienen al menos tres partes lógicas separadas, cada una ubicada en un servidor físico separado. Cada parte es responsable de la funcionalidad específica. Cuando se utiliza un enfoque de diseño en capas, se despliega una capa en un nivel si más de un servicio o aplicación depende de la funcionalidad expuesta por la capa.

Los principales beneficios de la arquitectura en N-capas/3-capas son:

***Mantenibilidad****.* Debido a que cada nivel es independiente de los otros niveles, las actualizaciones o cambios se pueden llevar a cabo sin afectar la aplicación en su conjunto.

***Escalabilidad****.* Dado que los niveles se basan en el despliegue de capas, la ampliación de una aplicación es razonablemente sencilla.

***Flexibilidad****.* Debido a que cada nivel puede ser administrado o escalado de forma independiente, la flexibilidad se incrementa.

***Disponibilidad****.* Las aplicaciones pueden aprovechar la arquitectura modular de sistemas habilitadores que utilizan componentes fácilmente escalables, lo que aumenta la disponibilidad.

Como un ejemplo, la capa de presentación no debe almacenar datos confidenciales, mientras que esto puede almacenarse en las capas de datos y de negocio. Considere la posibilidad de utilizar sólo tres niveles si está desarrollando una aplicación de intranet donde todos los servidores están ubicados dentro de la red privada; O una aplicación de Internet en la que los requisitos de seguridad no restringen el despliegue de la lógica de negocios en el servidor Web o en la Web que se enfrenta al público. Considere la posibilidad de utilizar más de tres capas si los requisitos de seguridad dictaminan que la lógica empresarial no se puede implementar en la red perimetral o que la aplicación utiliza mucho los recursos y desea descargarla a otro servidor.

**2.3.4 Arquitectura de Micro-servicios (MSA)**

La arquitectura de micro-servicio, es un método de desarrollo de software que ha crecido en popularidad en los últimos años. Para muchos desarrolladores se ha convertido en la forma preferida de crear aplicaciones empresariales. Gracias a su escalabilidad, este método arquitectónico se considera especialmente ideal cuando se tiene que habilitar el soporte para una amplia gama de plataformas y dispositivos, que abarcan la web, el móvil, Internet de las cosas y los portátiles, o simplemente cuando no está seguro de qué tipo de dispositivos Usted tendrá que apoyar en un futuro cada vez más nublado.

La idea detrás de los micro-servicios es que algunos tipos de aplicaciones se vuelven más fáciles de construir y mantener cuando se dividen en piezas más pequeñas y componibles que trabajan juntas. Cada componente se desarrolla por separado, y la aplicación es entonces simplemente la suma de sus componentes constituyentes. Esto es en contraste con una aplicación tradicional, "monolítica", que es todo desarrollado en una sola pieza.

Si bien no existe una definición estándar y formal de micro-servicios, existen ciertas características que nos ayudan a identificar el estilo. Esencialmente, la arquitectura de micro-servicios es un método de desarrollo de aplicaciones de software como un conjunto de servicios modulares, desplegables de forma independiente, en los que cada servicio ejecuta un proceso único y se comunica a través de un mecanismo ligero y bien definido para servir una meta de negocio.

El modo en que los servicios se comunican entre sí depende de los requisitos de su aplicación, pero muchos desarrolladores usan HTTP/REST con JSON. REST (Representational State Transfer) es un método de integración útil debido a su comparativamente menor complejidad sobre otros protocolos.

Para empezar a entender la arquitectura micro-servicios, ayuda a considerar su opuesto: el estilo arquitectónico monolítico. A diferencia de los micro-servicios, la aplicación monolítica siempre se construye como una unidad única y autónoma. En un modelo cliente-servidor, la aplicación del lado del servidor es un monolito que maneja las solicitudes HTTP, ejecuta la lógica y recupera/actualiza los datos en la base de datos subyacente. El problema con una arquitectura monolítica, sin embargo, es que todos los ciclos de cambio suelen terminar ligados entre sí. Una modificación realizada en una pequeña sección de una aplicación puede requerir la creación e implementación de una versión totalmente nueva. Si necesita escalar funciones específicas de una aplicación, puede que tenga que escalar la aplicación completa en lugar de sólo los componentes deseados. Aquí es donde la creación de micro-servicios puede venir al rescate.

Hay muchas razones por las que este enfoque se considera una manera más fácil de desarrollar grandes aplicaciones, aplicaciones empresariales particulares y varios tipos de software como servicio entregado a través de Internet.

Si está desarrollando una aplicación empresarial de servidor. Debe soportar una variedad de clientes diferentes, incluyendo navegadores de escritorio, navegadores móviles y aplicaciones móviles nativas. La aplicación también puede exponer una API para que terceras partes consuman. También podría integrarse con otras aplicaciones a través de servicios web o un intermediario de mensajes. La aplicación gestiona las solicitudes (solicitudes y mensajes HTTP) mediante la ejecución de la lógica empresarial; Acceder a una base de datos; Intercambiar mensajes con otros sistemas; Y devolver una respuesta HTML / JSON / XML.

**2.3.5 Justificación de la arquitectura**

Habiendo analizado las distintas tecnologías, se llegó a la conclusión que se va a usar la arquitectura de micro-servicios, debido a que es una de las arquitecturas modernas que se está utilizando en desarrollo de software, tiene características de ejecución rápida, además, puede ser utilizada en clientes como los dispositivos móviles.

Así mismo, los servicios se utiliza para responder a eventos; usualmente a las acciones del usuario, e invocar peticiones a la capa de datos. Por otro lado, el sistema también puede operar con más de un cliente a la vez, haciendo que el tiempo de desarrollo se reduzca considerablemente. Sin embargo, por razones de facilidad de mantenimiento y una estructuración eficiente del código, la arquitectura de micro-servicio ha sido dividida en N-Capas. Cabe resaltar que en la arquitectura N-Capas, cualquier capa superior contiene solamente la declaración de métodos para comunicarse con alguna con capa inferior a través de una interface.

**2.4 Aplicaciones web**

La aparición de los servicios web a principios de los 90’s, supuso una verdadera revolución en el campo de la informática y las telecomunicaciones. Con la integración de este nuevo servicio, Internet inicio una rápida transición hacia el ámbito empresarial y supuso un enorme impulso al crecimiento de la red. Lo que había surgido en plena guerra fría como un proyecto militar y que posteriormente fue dirigiéndose hacia el ámbito científico y académico, se empezó a convertir en un perfecto "escaparate virtual" en el que las empresas pudieran ofrecer sus productos y servicios rompiendo barreras geográficas y de comunicaciones [14]. En la actualidad las empresas no ven únicamente el servicio web como un mero escaparate o medio publicitario de enorme difusión. Internet, y en particular el servicio web, abre a las empresas enormes posibilidades. La utilización de tecnologías web permite agilizar los procesos, mejorar la productividad y aumentar la eficacia, además de abrir las puertas a nuevas formas de negocio en el mercado global que facilita Internet.

**2.4.1 Arquitectura Cliente/Servidor**

Cliente/Servidor es una arquitectura de red en la que cada ordenador o proceso en la red es cliente o servidor. Normalmente, los servidores son ordenadores potentes dedicados a gestionar unidades de disco , tráfico de red, datos o incluso aplicaciones, mientras que los clientes son máquinas menos potentes y usan los recursos que ofrecen los servidores.

Según [15] un sistema Cliente/Servidor suele presentar las siguientes características:

* Una combinación de la parte cliente (*front-end*) que interactúa con el usuario y la parte servidor (*back-end*) que interactúa con los recursos compartidos (bases de datos).
* La parte cliente y servidor tienen diferentes necesidades de recursos a la hora de ejecutarse: velocidad de procesador, memoria, velocidad y capacidad de los discos duros, dispositivos de entrada/salida, etc.
* El entorno suele ser heterogéneo y multivendedor. El hardware y sistema operativo del cliente y el servidor suelen diferir.
* Normalmente la parte cliente se implementa haciendo uso de una interfaz gráfica de usuario, que permite la introducción de datos a través de teclado, ratón, lápiz, óptico, etc.

**Figura 2.4**

**Arquitectura cliente - servidor**

[](https://github.com/wolf-mtwo/lagash-documentation/blob/dev/anexos/figura-4.png)

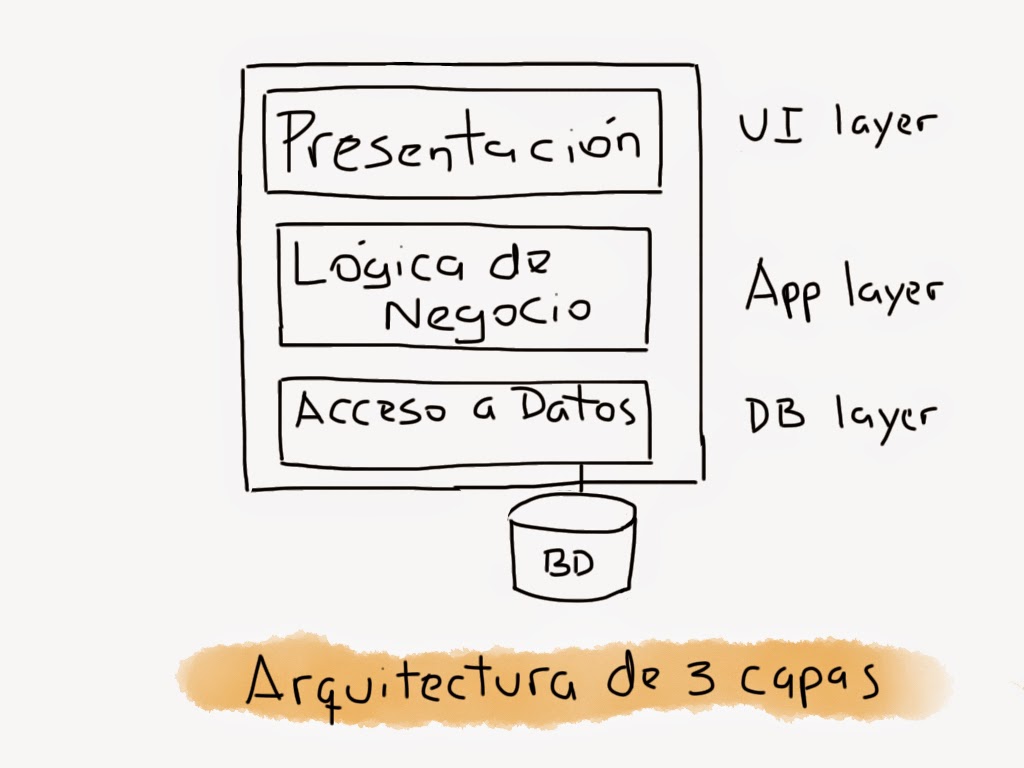
***Fuente:****[Elaboración propia]*

La arquitectura Cliente/Servidor nos permite la separación de funciones en tres niveles, tal como muestra la siguiente *figura 2.5*.

* **Lógica de presentación.** La presentación de los datos en una función independiente del resto.
* **Lógica de negocio.** Los flujos de trabajo pueden cambiarse según las necesidades existentes de un procesador a otro.
* **Lógica de datos.** La gestión de los datos debe ser independiente para poder ser distribuida según las necesidades de la empresa en cada momento.

**Figura 2.5**

**Separación de funciones en tres niveles**

[](https://github.com/wolf-mtwo/lagash-documentation/blob/dev/anexos/figura-5.jpg)

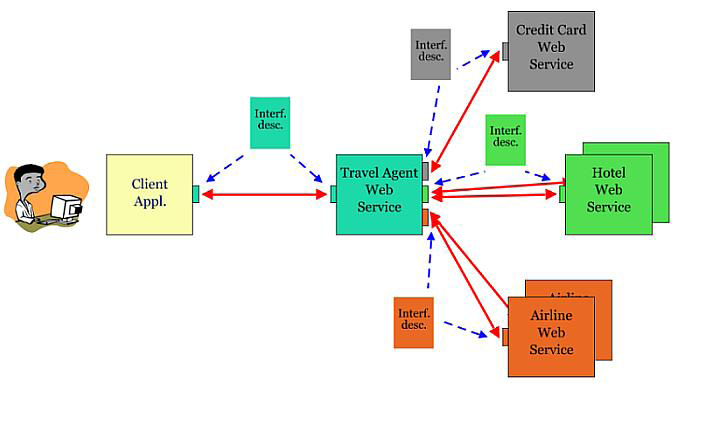
***Fuente:****[Elaboración propia]*

**2.4.2 Servicios web**

Los servicios web son una de las últimas nuevas tecnologías que han llegado a la informática. Servicio web representa un recurso de información o un proceso de negocio, al que puede acceder otra aplicación a través de la web y con el cual se puede comunicar a través de protocolos estándares de internet. La particularidad que tienen los servicios web es que están diseñados para permitir la comunicación de una aplicación con otra, sin intervención humana. Según [16] existen diferentes estándares que especifican la forma de definir servicios web sobre HTTP, de forma que a la hora de desarrollar servicios web, a continuación se verá los dos principales estándares para definir servicios web.

**Figura 2.6**

**Ejemplo del trabajo del Servicio Web**

[](https://github.com/wolf-mtwo/lagash-documentation/blob/dev/anexos/figura-6.png)

***Fuente:****[16, fig. 5.1]*

**2.4.2.1 Justificación del servicio web e intercambio de datos**

REST es un estilo arquitectónico, una forma de exponer servicios o funcionalidades de un sistema, sea desarrollado en cualquier lenguaje de programación. Pretende exponer y separar ciertas partes de las funcionalidades de una forma adecuada con GET y POST. La función es ordenar, acomodar y hacer más sencilla la programación para luego poder darle un buen mantenimiento después de terminar el desarrollo.

REST son más pequeñas, más concisas y no contienen toda la información o metadatos como lo hacen las solicitudes de SOAP y normalmente no están validadas por el cliente antes de enviar por esto son pequeños y considerablemente más rápidas, a diferencia de SOAP, los datos devueltos por un servicio REST pueden estar en gran variedad de formatos. Se utilizará el formato de intercambio de datos JSON porque es más liviano que XML porque utiliza menos metadatos, es más pequeño y también pueden devolver archivos binarios, imágenes, etc. Y no solo así intercambio de datos en texto.

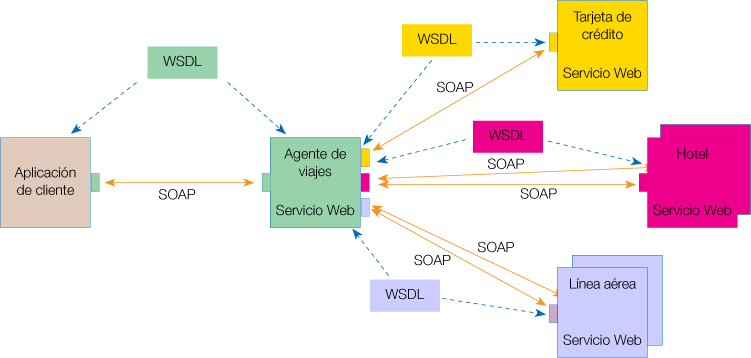
**2.4.2.2 SOAP (Simple Objet Access Protocol)**

Es un estándar del W3C que define como objetos remotos pueden comunicarse mediante el intercambio de XML. La idea básica es que en la comunicación hay dos partes (cliente y servidor), una de las cuales (el servidor) presta una serie de servicios que son consumidos por la otra (cliente). Lo más habitual es que el servidor haga pública la especificación de sus servicios mediante un documento WSDL (Web Service Description Lenguage) que se puede utilizar construyendo un cliente que invoque tales servicios.

Los servicios web SOAP están orientados a funcionalidad. El servidor implementa una serie de funcionalidades y le dice al mundo como pueden invocarse [16]. SOAP fue diseñado originalmente por varias empresas, entre ellas Microsoft, para permitir una gran comunicación entre objetos en los programas. En un sentido, la finalidad de SOAP fue el primero en definir una forma de usar objetos en Internet, con protocolos y formatos estándar, en lugar de los formatos binarios y los protocolos propietarios [17].

**Figura 2.7**

**Arquitectura SOAP**

[](https://github.com/wolf-mtwo/lagash-documentation/blob/dev/anexos/figura-7.png)

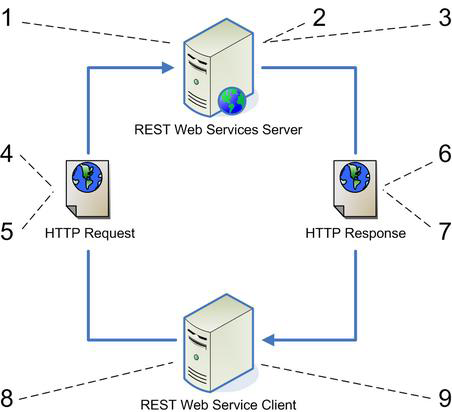
***Fuente:****[17, fig. 3.4]*

**2.4.2.3 REST (Representational State Transfer)**

Es un conjunto de técnicas orientadas a crear servicios web en los que se renuncia a la posibilidad de especificar la interfaz de los servicios de forma abstracta a cambio de contar con una convención que permite manejar la información mediante una serie de operaciones estándar. La convención utilizada no es otra que el protocolo HTTP. Según [17], los servicios REST están orientados a la manipulación de recursos. En un servicio REST, tenemos una URL por cada recurso que se gestiona, y que se realiza una tarea diferente sobre dicho recurso en función del método HTTP que se vaya a utilizar.

**Figura 2.8**

**Estructura REST Fuente**

[](https://github.com/wolf-mtwo/lagash-documentation/blob/dev/anexos/figura-8.png)

***Fuente:****[17, fig. 4.1]*

Los principios de REST son los siguientes:

* El estado de la aplicación y la funcionalidad están divididos en recursos.
* Todos los recursos son accesibles de forma única mediante una síntesis común en forma de enlaces a hipermedios.
* Todos los recursos comparten una interfaz común para la transferencia de estado entre el cliente y el recurso, consistente en un conjunto estricto de operaciones bien definidas.
* Un protocolo Cliente/Servidor que no guarde el estado, cuyos resultados puedan almacenarse en una memoria cache, y que este claramente separado en capas.

**2.4.3 Intercambio de datos**

Para intercambiar datos entre el sistema web y la aplicación Android, se utilizará JSON, porque es liviano, ya que sus datos se almacenan en vectores y registros, mientras que XML almacena los datos en árboles, de manera que resulta más difícil familiarizar a los lenguajes orientados a objetos. De esta manera, JSON hace más sencillo importar datos a diferentes lenguajes de programación.

**2.4.3.1 Lenguaje de marcado extensible (XML)**

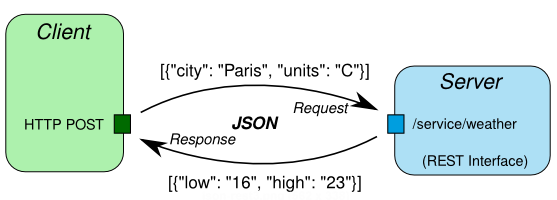
XML es conocido también como un archivo de configuración, también es utilizado para el mapeo de base de datos implicado directamente en el desarrollo de software, "fue desarrollado inicialmente para internet, pero gracias a las distintas DTD a las que puede asociarse, puede servir en la actualidad para distintas aplicaciones como formato de intercambio o como sistema de almacenamiento de datos" [18]. Es un lenguaje muy similar a HTML pero su función principal es describir datos sin mostrarlos como es el caso de HTML. XML es un formato que permite la lectura de datos a través de diferentes aplicaciones.

**2.4.3.2 Notación de objetos de JavaScript (JSON)**

Se define como archivo de datos, es utilizado como archivo de configuración, JSON (JavaScript Object Notation) es un formato ligero de intercambio de datos. Para las máquinas es simple interpretarlo y generarlo. JSON es un formato utilizado en JavaScript para serializar datos. Se lo usa como formato de intercambio de datos "liviano" (en comparación con XML), y tiene la particularidad que los datos en este formato son leídos de manera directa en JS. Se basa en la construcción de una lista ordenada de valores, listas de objetos, que pueden incluir a su vez tablas hash, objetos con una colección de pares nombre/valor [19].

**Figura 2.9**

**Sintaxis y estructura de JSON Fuente**

[](https://github.com/wolf-mtwo/lagash-documentation/blob/dev/anexos/figura-9.png)

***Fuente:****[19, fig. 2.3]*

**2.5 Tipos de aplicaciones móviles**

En el desarrollo de aplicaciones para dispositivos móviles existen: la aplicación web móvil, aplicación nativa y la aplicación hibrida, a continuación se describirán cada uno de ellos.

**2.5.1 Aplicaciones nativas**

Este tipo de aplicaciones están hechas para ejecutarse en un dispositivo y sistema operativo específico. Así, la mayor parte de las aplicaciones descargadas de la App Store de Apple son aplicaciones que sólo van a correr sobre Iphone e Ipad. Este tipo de aplicaciones se crean con distintos tipos de lenguajes. Las desarrolladas para iOS lo hacen con los lenguajes: Objective C, C, o C++. Las aplicaciones desarrolladas para el sistema operativo Android lo hacen con lenguaje Java. Este tipo de aplicaciones corren de forma más eficiente sobre estos dispositivos ya que sus componentes están diseñados de forma específica para este sistema operativo. Además, este tipo de aplicaciones pueden emplear todos los sensores y elementos del teléfono: camara, gps, acelerómetro, agenda, etc. Esta es una diferencia fundamental con respecto a las aplicaciones web [20]. En la figura siguiente vemos los principales entornos, en función del sistema operativo del dispositivo:

**Figura 2.10**

**Sistemas operativos de dispositivos móviles**

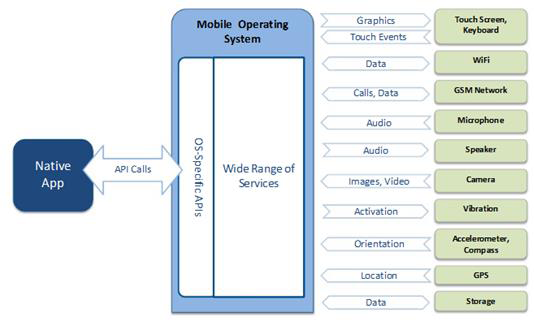
[](https://github.com/wolf-mtwo/lagash-documentation/blob/dev/anexos/figura-10.png)

***Fuente:****[21]*

Las aplicaciones nativas tienen pleno acceso a la API del sistema operativo en el que se desarrollan. Su código fuente hace que las llamadas a la API de sistema operativo específico para acceder a los diferentes servicios disponibles en la plataforma. Esto variará según la plataforma, pero todas las aplicaciones de forma nativa desarrollados tienen acceso completo a las funciones previstas. Esto incluye funciones tales como el almacenamiento de datos, la ubicación, la cámara, altavoz, micrófono, y otros.

**Figura 2.11**

**Interacción aplicaciones nativas**

[](https://github.com/wolf-mtwo/lagash-documentation/blob/dev/anexos/figura-11.png)

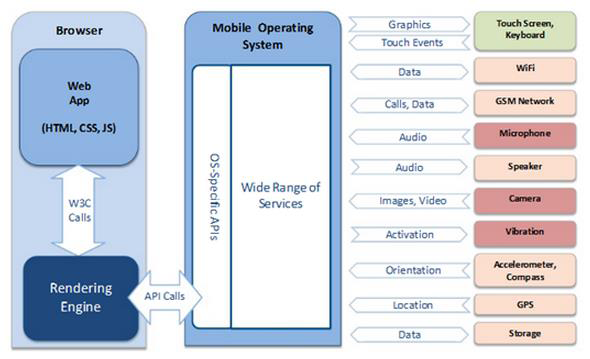
***Fuente:****[21]*

**2.5.2 Aplicaciones web móviles**

Las aplicaciones web móviles, a diferencia de las aplicaciones nativas, se ejecutan dentro del navegador del teléfono. La principal relevancia de utilizar aplicaciones web es que son multiplataforma, se podrá crear una aplicación una vez y distribuirlo hacia todas las plataformas y con algunos ajustes para el correcto funcionamiento en todos los dispositivos pero no implicará un desarrollo por cada plataforma. La aplicación web es aquella que se encuentra instalada en un servidor tipo web o un browser y necesita de él para ejecutarse, dicho navegador debe de ser compatible con las tecnologías con las cuales se realiza el aplicativo WEB. Estas aplicaciones están desarrolladas con HTML, CSS y Javascript. [22]

**Figura 2.12**

**Interacción aplicaciones web móviles**

[](https://github.com/wolf-mtwo/lagash-documentation/blob/dev/anexos/figura-12.png)

***Fuente:****[21]*

A pesar de los grandes avances de HTML5 al no estar haciendo un desarrollo nativo con las SDKs y APIs disponibles para cada plataforma se está perdiendo una parte importante de la funcionalidad del dispositivo así como en apariencia, siendo complicado que la interfaz se sienta tan ligada al teléfono como en casos nativos. También puede ser complicado lograr un solo diseño que se adapte perfectamente a todos los dispositivos a la vez.

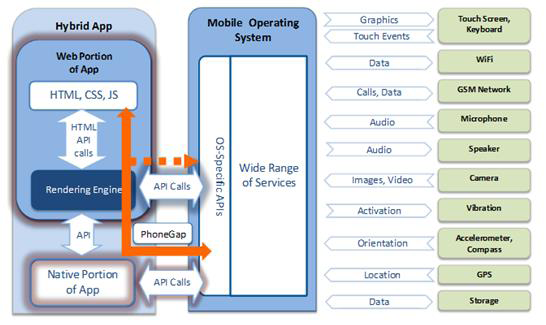
**2.5.3 Aplicaciones híbridas**

Las aplicaciones híbridas asocian lo mejor de los dos anteriores modelos. Este tipo de aplicaciones permite el uso de tecnologías multiplataforma como HTML, Javascript y CSS pero permiten acceder a buena parte de los dispositivos y sensores del teléfono. Buena parte de la infraestructura es tipo web y la comunicación con los elementos del teléfono se hace mediante comunicadores tales como phonegap Un buen ejemplo de aplicaciones híbridas es Facebook. Se descarga de la app store y cuenta con todas las características de una aplicación nativa pero requiere ser actualizada ocasionalmente. [20]

El proceso de desarrollo para este tipo de aplicaciones es algo más complicado. Al igual que para las aplicaciones nativas, el código una vez creado se compila a un ejecutable. Además, también como en las aplicaciones Web se genera código HTML, CSS y Javascript a ejecutar en un navegador. Ambos códigos se compilan para ser subidos mediante un paquete distribuible a la app store.

**Figura 2.13**

**Interacción aplicaciones hibridas**

[](https://github.com/wolf-mtwo/lagash-documentation/blob/dev/anexos/figura-13.png)

***Fuente:****[21]*

**2.5.4 Elección de la aplicación móvil**

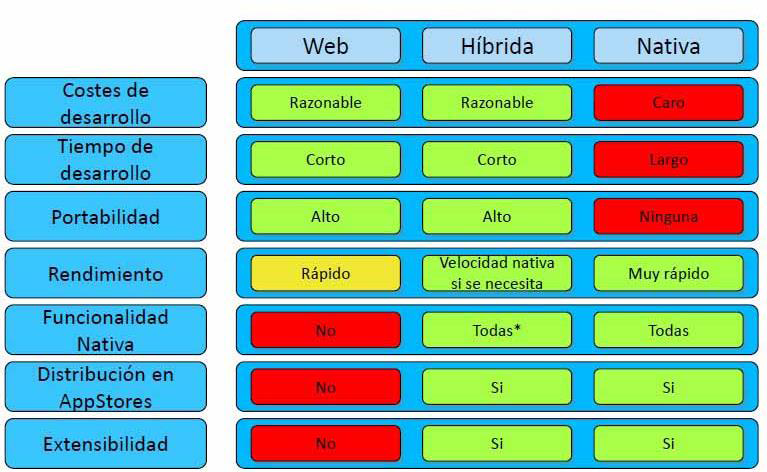
Para el desarrollo de la aplicación de este proyecto se utilizará la aplicación nativa, ya que la conexión a internet no es necesaria y no requiere una constante actualización de la aplicación.

Cabe destacar que las aplicaciones nativas tienen acceso total a las utilidades del sistema operativo del dispositivo: Dispositivos de almacenamiento, cámara, gps, acelerómetro, etc.

Además del total acceso a los elementos del teléfono las aplicaciones nativas no requieren de conexión web para ser ejecutadas. Es importante destacar que las aplicaciones nativas tendrán mucha más visibilidad ya que se distribuyen a través de la app store de los fabricantes. [20]

**Figura 2.14**

**Comparación de tipo de aplicaciones**

[](https://github.com/wolf-mtwo/lagash-documentation/blob/dev/anexos/figura-14.png)

***Fuente:****[21]*

**2.6 Indexación única**

Existen varias maneras de identificación que se tornaron populares en la actualidad y podemos mencionar entre ellos: UUID (Universally unique identifier) [7], y QR (Quick response codes) [9]. Cada una estas formas de identificación se utiliza de forma separada, podemos mencionar que un UUID es una cadena de caracteres y en cambio un QR es un contenedor de datos que se vuelve único en base a su contenido.

**2.6.1 Identificador único universal (UUID)**

Un UUID es un identificador que es único a lo largo entre espacio y tiempo... para identificar de forma fiable objetos persistentes a través de una red. [10] entre sus características se puede mencionar que un UUID es una cadena de 32 caracteres alfanuméricos en minúscula con una forma de 8-4-4-4-12 y un total de 36 caracteres. un UUID tendría un parecido a este identificador, ejemplo: 00000000-0000-0000-0000-000000000000, entre las posibilidades de combinaciones de los caracteres alfanuméricos es imposible poder duplicar la generación de uno igual.

**2.6.2 Códigos de respuesta rápida (QR)**

Los códigos QR han ganado popularidad en los últimos años por su capacidad de almacenaje de datos y son mucho más efectivo que el código de barras, un código QR puede contener varios tipos de datos mensajes, texto, URL, geolocalización. La capacidad de un QR es muy variado, pero con la ayuda de los UUID los QR pasarían a convertirse en entidades únicas y perfectos a la hora de identificar un recurso bibliográfico, y sin tener que estar asociado a ningún sistema.

**2.7 Lenguajes de programación**

Un lenguaje de programación es un lenguaje formal diseñado para realizar procesos que pueden ser llevados a cabo por máquinas como las computadoras. Pueden usarse para crear programas que controlen el comportamiento físico y lógico de una máquina, para expresar algoritmos con precisión, o como modo de comunicación humana. Está formado por un conjunto de símbolos y reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos y expresiones. Al proceso por el cual se escribe, se prueba, se depura, se compila (de ser necesario) y se mantiene el código fuente de un programa informático se le llama programación.

**2.7.1 C Sharp (C#)**

C# es un lenguaje elegante, con seguridad de tipos y orientado a objetos, que permite a los desarrolladores crear una gran variedad de aplicaciones seguras y sólidas que se ejecutan en .NET Framework .NET. Puede usar C# para crear aplicaciones cliente de Windows, servicios web XML, componentes distribuidos, aplicaciones cliente-servidor, aplicaciones de base de datos y muchas, muchas más cosas.

**2.7.2 Java**

Java es un lenguaje de programación y una plataforma informática comercializada por primera vez en 1995 por Sun Microsystems. Hay muchas aplicaciones y sitios web que no funcionarán a menos que tenga Java instalado y cada día se crean más. Java es rápido, seguro y fiable. Desde portátiles hasta centros de datos, desde consolas para juegos hasta súper computadoras, desde teléfonos móviles hasta Internet, Java está en todas partes.

**2.7.3 JavaScript**

Javascript es un lenguaje con muchas posibilidades, utilizado para crear pequeños programas que luego son insertados en una página web y en programas más grandes, orientados a objetos mucho más complejos. Con Javascript podemos crear diferentes efectos e interactuar con nuestros usuarios.

Este lenguaje posee varias características, entre ellas podemos mencionar que es un lenguaje basado en acciones que posee menos restricciones. Además, es un lenguaje que utiliza Windows y sistemas X-Windows, gran parte de la programación en este lenguaje está centrada en describir objetos, escribir funciones que respondan a movimientos del mouse, aperturas, utilización de teclas, cargas de páginas entre otros.

**2.7.4 Conclusión**

La decisión de qué lenguaje de programación usar a la hora del desarrollo aplicaciones web, siempre será tema de controversia. Las diferencias entre estos, generalmente se encuentran en la plataforma que requieran. Por ello podrá observarse que la elección del lenguaje, tenderá más hacia el gusto del usuario, que al resultado de evaluar el rendimiento de las herramientas.

**2.7.5 Elección del lenguaje de programación**

La arquitectura de microservicios sin duda tiene muchas ventajas. En particular permite ser agnóstico con los lenguajes de programación. Aun así los tres lenguajes mencionados serán utilizados dentro el sistema ya que cada una cuenta con gran potencial en las áreas que se realizara su implementación.

**2.8 Calidad del software**

Es software que se aplica de manera útil que crea un producto, que proporciona valor medible a quienes lo producen y a quienes lo utilizan.

**2.8.1 Pruebas unitarias**

Las pruebas unitarias se utilizan para ejecutar otro código fuente llamando directamente a los métodos de una clase, pasando los parámetros apropiado, si incluye instrucciones ASSERT, éstas pueden probar los valores que se generan con respecto a los valores esperados. Los métodos de pruebas unitarias residen en clases Test, que se almacenan en archivos de código fuente [25].

Unit Test es un trozo de código desarrollado con el único objetivo de verificar que una rutina o función de nuestro código está funcionando según se espera [26]. Para asegurar la calidad del código entregado se realizan pruebas unitarias, es la mejor forma de detectar errores en el desarrollo [25].

Para que una prueba unitaria funciones correctamente se debe seguir los siguientes requisitos.

* **Completas.** Deben cubrir la mayor cantidad de código.
* **Repetibles o reutilizables.** No se deben crear pruebas que solo puedan ser ejecutadas una sola vez.
* **Independientes.** La ejecución de una prueba no debe afectar a la ejecución de otra.
* **Profesionales.** Las pruebas deben ser consideradas igual que el código, con la misma profesionalidad y documentación.

El objetivo de las pruebas unitarias es aislar cada parte del programa y mostrar que las partes individuales del código son correctas. Por la estructura de la arquitectura del presente proyecto la facilidad de uso de las pruebas unitarias es más aplicable ya que a nivel del cliente se realizaran Unit Testing.

**2.8.2 Usabilidad**

La Usabilidad es la medida de la calidad de la experiencia que tiene un usuario cuando interactúa con un producto o sistema. Esto se mide a través del estudio de la relación que se produce entre las herramientas y quienes las utilizan, para determinar la eficiencia en el uso de los diferentes elementos ofrecidos en las pantallas y la efectividad en el cumplimiento de las tareas que se pueden llevar a cabo a través de ellas.

Otra definición es la que entrega el académico Yusef Hassan (Universidad de Granada) al indicar que "la usabilidad es la disciplina que estudia la forma de diseñar Sitios Web para que los usuarios puedan interactuar con ellos de la forma más fácil, cómoda e intuitiva posible" y agregar que "la mejor forma de crear un sitio web usable es realizando un diseño centrado en el usuario, diseñando para y por el usuario, en contraposición a lo que podrá ser un diseño centrado en la tecnología o uno centrado en la creatividad u originalidad" [25].

**2.8.3 Seguridad del sistema**

Los principios de seguridad de la información están basadas en confidencialidad, integridad, disponibilidad y autenticidad. Los mecanismos básicos de seguridad, según la norma ISO 17799 es la preservación de la confidencialidad, integridad y disponibilidad de información, también toma en cuenta las propiedades de privacidad, identificación, control de accesos y la fidelidad [27].

Para lograr sus objetivos la seguridad informática se fundamenta en tres principios, que debe cumplir todo sistema informático [23]:

**Confidencialidad.** Se refiere a la privacidad de los elementos de información almacenados y procesados en un sistema informático, Basándose en este principio, las herramientas de seguridad informática deben proteger el sistema de invasiones y accesos por parte de personas o programas no autorizados. Este principio es particularmente importante en sistemas distribuidos, es decir, aquellos en los que los usuarios, computadores y datos residen en localidades diferentes, pero están física y lógicamente interconectados.

En el presente proyecto se tomara en cuenta la gestión de usuarios que incluye normas de confidencialidad lo cual se describe a continuación.

* **Tiempo de sesión abierta.** En el sistema se tomara en cuenta el tiempo de espera en acciones del usuario con el sistema.
* **Fortaleza de la contraseña.** La contraseña debe tener un número mínimo de caracteres con símbolos.
* **Intentos continuos permitidos.** El número de intentos para el ingreso al sistema es de tres, posteriormente se bloqueara el sistema.
* **Encriptación de los datos.** En el proceso de autentificación los datos que se ingresen como usuario y contraseña serán protegido.

**Integridad.** Se refiere a la validez y consistencia de los elementos de información almacenados y procesador en un sistema informático. Basándose en este principio, las herramientas de seguridad informática deben asegurar que los procesos de actualización estén bien sincronizados y no se dupliquen, de forma que todos los elementos del sistema manipulen adecuadamente los mismos datos. Este principio es importante en sistemas descentralizados, es decir, aquellos en los que diferentes usuarios, computadores y procesos comparten la misma información.

Al realizar acciones, el sistema garantiza que los datos fueron procesados de manera correcta para ello se realizara pruebas unitarias de manera constante, ya que la metodología seleccionada para el desarrollo del sistema apoya.

**Disponibilidad.** Se refiere a la continuidad de acceso a los elementos de información almacenados y procesados en un sistema informático, basándose en este principio, las herramientas de seguridad informática deber reforzar la permanencia del sistema informático, en condiciones de actividad adecuadas para que los usuarios accedan a los datos con la frecuencia y dedicación que requieran.

Este principio es importante en sistemas informáticos cuyo compromiso con el usuario, es prestar servicio permanente.

**Autenticidad.** Es el proceso que se realiza para la identificación del usuario, en el desarrollo del proyecto se tomaran en cuenta los siguientes métodos de autentificación para la verificación de identidad.

* **Autentificación de usuario.** Previene el ingreso a personas no autorizadas, el usuario se da conocer mediante un nombre de usuario y su respectiva contraseña
* **Permiso por roles.** Restringe al sistema de usuario no autorizados para ciertas funcionalidades del sistema, aplicando en ella la integración de la información.
* **Modalidad de accesos.** El ingreso al sistema con ciertos accesos permitidos por un rol, teniendo la capacidad de desactivar un usuario en caso de contingencias a prepararse.
* **Monitoreo del control de acceso.** El control de acceso es controlado automáticamente, obteniendo los datos necesarios de las acciones realzadas. Se ve conveniente desarrollar un control de seguridad en diferentes aspectos del sistema para prevenir posibles ataques. Los datos que se muestran en la siguiente tabla son considerados para un soporte de seguridad del sistema.

**REFERENCIAS BIBLIOGRÁFICAS**

1. Y. Gascón, *Revisión de conceptos básicos Modelado de Negocios,* Disponible en: <http://es.slideshare.net/yamilagascon/revisin-de-conceptos-bsicos-modelado-de-negocios>
2. J. Montilva, *Modelo de Negocio,* Disponible en: <http://es.slideshare.net/RaulIram/tarea-16614847>
3. Ian Sommerville (2005) *Ingeniería del Software,* 7th edición, Madrid, España, Pearson Educación.
4. R. S. Pressman, *Ingeniería del software: Un enfoque práctico,* 7th. Edición, The McGraw-HillCompanies.
5. Pilar Rodríguez Gonzales, *Estudio de la aplicación de metodologías ágiles para la evolución de productos software,* Master en tecnologías de la información. Universidad Politécnica De Madrid 2008
6. Roger S. Pressman, *Ingeniería de software,* en Ingeniería del software un enfoque práctico.7ª 1edición, México: Mexicana, 2010, Cap.1, pp. 10
7. UUID, *Universally unique identifier,* Disponible en: <https://en.wikipedia.org/wiki/Universally_unique_identifier>
8. Martin Fowler, *Patterns of Enterprise Application Architecture,* Disponible en: <ftp://ftp.heanet.ie/mirrors/sourceforge/w/we/webtune/Patterns%20of%20Enterprise%20Application%20Architecture.pdf>
9. QR, *Quick response codes,* Disponible en: <http://www.qrcode.com/en/about/>
10. UUID, *Definition,* Disponible en: <http://www.ietf.org/rfc/rfc4122.txt>
11. Marshall, Catherine & Rossman, Gretchen B. (1989). *Designing qualitative research,* Newbury Park, CA: Sage.
12. Mobile app services, *mobile and integration,* Disponible en: https://azure.microsoft.com/en-us/documentation/articles/app-service-mobile-value-prop/
13. patterns & practices Developer Center, *What is Software Architecture?,* Disponible en: <https://msdn.microsoft.com/en-us/library/ff650706.aspx>
14. Angel Cobo, *"Internet y la programación de ordenadores" en PHP y MySQL,* 1ª Edición. España: Díaz de Santos, 2005. Cap. Nº 1, 12 pp. 13-20.
15. Sergio Lujan Mora, *"¿Que es una aplicación web?" en Programación en Internet,* 1ª Edición. España: Club Universitario, 2010.
16. Duncan Mackenzie, *"Servicio web XML" en Aprendiendo Visual Basic.Net Lecciones Avanzadas,* 1ª Edición. México: Pearson Educación.
17. *Manual de desarrollo web,* 1ª Edición, ISBN, México, 2010.
18. Nicholas C. Zakas, *"Ajax" en Alto rendimiento de JavaScript,* 1ª Edición. EE. UU.: O'Reylly Media, Inc., 2010. Cap. Nº 7, 8 pp, pag. 134.
19. Sebastian Bassi, *"JSON" en serialización de datos,* 4ª Edición. Argentina: Gemes Digitales, 2013. Cap 5, 4 pp, 30-31.
20. Geospatial, *Tipos de aplicaciones móviles,* Disponible en: <http://geospatialtraininges.com/recursos-gratuitos/tipos-de-aplicaciones-moviles/>
21. GeoChalkboard, *Anatomy of a Hybrid Mobile GIS Application,* Disponible en: <http://www.geospatialtraining.com/blog/index.php/anatomy-of-a-hybrid-mobile-gis-application/>
22. Yeicy Juliana Molina Rivera, Jonathan Sandoval Cardona, Santiago Alberto Toledo Franco, *Sistema Operativo Android: Características y Funcionalidad para Dispositivos Móviles Tesis en Licenciatura en Ingeniería de Sistemas y Computación,*Universidad Tecnológica de Pereira.
23. R. S. Pressman, *Software y la Ingeniería de Software, Ingeniería de Software (un enfoque práctico),* ISBM 978-607-15-0314-5, Libro de edición México, 2010.
24. J.Rodríguez, *Pruebas Unitarias, 2011,* Disponible en: <http://es.slideshare.net/ocomur/pruebas-unitarias-9368721?next_slideshow=1>
25. Policarpio, *Modelo incremental,* Disponible en: <http://phpwebquest.org/cursocep/webquest/soporte_tabbed_w.php?id_actividad=18518&id_pagina=1>
26. I, Tecnología, *Ingeniería de Software,* Libro de edición E.E.U.U., Marzo 2009.
27. E. Pérez, Seguridad y auditoria informática, ISO 17799 , noviembre 27 , 2011, Disponible en: <http://es.slideshare.net/luchoapazam1/seguridad-y-auditoria-informatica-iso-17799>