UNIVERSIDAD MILITAR
NUEVA GRANADA

DESARROLLO DE UNA ARQUITECTURA DE SOFTWARE PARA GESTIÓN DE INFORMACIÓN NO ESTRUCTURADA.

Daniel Camilo Daza Díaz

Alex Mauricio Florez Mendoza

UNIVERSIDAD MILITAR NUEVA GRANADA FACULTAD DE INGENIERÍA
PROGRAMA DE INGENIERÍA EN MULTIMEDIA
BOGOTA D.C.
2013
UNIVERSIDAD MILITAR NUEVA GRANADA

DESARROLLO DE UNA ARQUITECTURA DE SOFTWARE PARA GESTIÓN DE INFORMACIÓN NO ESTRUCTURADA

Daniel Camilo Daza Díaz

Alex Mauricio Florez Mendoza

Trabajo de grado presentado como requisito parcial para obtener el título de
INGENIERÍA EN MULTIMEDIA

Director
Eduard Leonardo Sierra

UNIVERSIDAD MILITAR NUEVA GRANADA FACULTAD DE INGENIERÍA
PROGRAMA DE INGENIERÍA EN MULTIMEDIA
BOGOTA D.C.
2013

Nota de aceptación

Director

Jurado

Jurado

Comentarios

Agradecimientos

Este trabajo es la culminación de una etapa formativa bastante productiva e importante en mi desarrollo integral, este objetivo conseguido es fruto de la colaboración de mi familia que ha sido un pilar en mi formación personal y gran apoyo en mi formación educativa, agradezco al ingeniero Eduard Leonardo Sierra, por acompañarme en este proceso y contribuir para conseguir el resultado que se presenta en este documento.

Alex Mauricio Florez Mendoza

Hoy luego de haber realizado este proyecto deseo agradecer primero a mi compañero de opción de grado por todo el trabajo y esfuerzo que puso en éste. También quiero agradecer a mi familia y a la de mi compañero por todo el apoyo en estos largos meses de trabajo. La ayuda del tutor Eduard Sierra fue de gran ayuda, porque nos ayudó a potenciar nuestras debilidades, para sacar este proyecto adelante y por ultimo pero no menos importante a Andrés Carreño quién es el dueño de la idea y nos la cedió para que este proyecto fuera una realidad.

Daniel Camilo Daza Díaz

CONTENIDO

LISTA DE FIGURAS 7

RESUMEN 9

INTRODUCCIÓN 10

2. RESUMEN DE LA PROPUESTA 12

2.1 OBJETIVOS 12

2.2 METODOLOGIA 13

3. MARCO TEÓRICO 15

3.1 GESTIÓN DE LA INFORMACIÓN 15
3.1.1 Bases de Datos 16
3.1.2 Dbms de Uso General y Especial 16
3.1.3 Modelo de Bases de Datos 17
3.1.4 SQL 18
3.1.5 No SQL 18

3.2 ARQUITECTURA CLIENTE - SERVIDOR 19

3.3 PROTOCOLOS DE COMUNICACIÓN 19
3.3.1 HTTP 19
3.3.2 REST 20
3.3.2.1 API REST 20

3.4 ARQUITECTURAS DE SOFTWARE 21
3.4.1 Modelo Vista Controlador - MVC 21

3.5 SERVIDOR WEB 22
3.5.1 Node.Js 22
3.5.1.1 Express.Js 22

4. ARQUITECTURA DE SOFTWARE 23

4.1 REQUERIMIENTOS DE LA ARQUITECTURA 23
4.1.1 Dominio del Problema 23
4.1.1.1 Análisis del dominio 23
4.1.1.2 Características de Uso 24
4.1.1.3 Características Funcionales 24
4.1.1.4 Características No Funcionales 24
4.1.1.5 Público Objetivo 25
4.1.2 Requerimientos del Cliente 25

4.1.2.1 Características Técnicas 25
4.1.3 Requerimientos del Servidor 25
4.1.3.1 Requerimientos funcionales 25
4.1.3.2 Requerimientos No funcionales 25
4.1.3.3 Características Técnicas 25

4.2 DISEÑO DE LA ARQUITECTURA 26
4.2.1 Arquitectura De Almacenamiento De Información 27
4.2.1.1 Base de Datos 27
4.2.1.2 Sistema de Comunicación con la Base de Datos 29
4.2.2 Diseño del Flujo de la Información 30
4.2.3 Modelo de la Aplicación 31
4.2.4 Diseño del Protocolo de Comunicación Sistema 31

5 EJEMPLO DE APLICACIÓN 35

5.1 REQUERIMIENTOS DEL SISTEMA 35
5.1.1 Características técnicas 35
5.1.2 Características de Uso 36
5.1.3 Características Funcionales 36

5.2 PERSONALIZACIÓN DE LA ARQUITECTURA 36
5.2.1 Personalización del Modelado de datos 37

5.3 DISEÑO 39

6 RESULTADOS Y ANALISIS 41

6.1 RESULTADOS DE DESEMPEÑO 41
6.1.1 Inserción de Datos 44
6.1.2 Búsqueda de Datos 45
6.1.3 Modificación de Datos 47
6.1.4 Eliminación de Datos 48

6.2 COMPARACIÒN DE RENDIMIENTO 49

6.3 ANALISIS 52

6.3.1 Clasificación del sistema cliente/servidor. 53

6.3.2 Arquitectura de Almacenamiento de Información 53

6.3.3 Modelo de la Aplicación 54
6.3.4 Protocolo de Comunicación 54

Anexos Digitales 57

Tablas 57

BIBLIOGRAFÍA 58

LISTA DE FIGURAS

2.1 Metodología e implementación, Diagrama de la metodología del proyecto.
4.1 Arquitectura Cliente/Servidor, Diagrama general de la arquitectura.
4.2 Estructura de almacenamiento, Colecciones internas en la base de datos.
4.3 Estructura de almacenamiento Interno de las colecciones en Mongo DB.
4.4 Modelado de datos, Estructuración de documentos.
5.1 Personalización de la Arquitectura.
5.2 Arquitectura general personalizada.
5.3 Diseño gráfico y flujo del ejemplo de aplicación.
6.1 Envío de petición, Petición enviada desde consola de comandos.
6.2 Envío de petición, Petición enviada desde un cliente web.
6.3 Envío de petición, Petición enviada desde un cliente Móvil.
6.4 Inserción de Texto, Tiempo de respuesta en el Cliente, Tiempo/ Peticiones.
6.5 Inserción de Imagen, Tiempo de respuesta en el Cliente, Tiempo/ Peticiones.
6.6 Búsqueda por texto, Tiempo de respuesta en el Cliente, Tiempo/ Peticiones.
6.7 Búsqueda por Geolocalización, Tiempo de respuesta en el Cliente, Tiempo/ Peticiones.
6.8 Modificación de Datos, Tiempo de respuesta en el Cliente, Tiempo/ Peticiones.
6.9 Eliminación de datos, Tiempo de respuesta en el Cliente, Tiempo/ Peticiones.
7.1 Comparación de Rendimiento en el Servidor, Tiempo de Respuesta.
7.2 Comparación de Rendimiento en el Servidor, Tiempo de Ejecución del Proceso.
7.3 Comparación de Rendimiento en el Cliente Móvil, Tiempo de Respuesta.

RESUMEN

En este documento se mostrara todo el planteamiento e implementación para el desarrollo de una arquitectura de software encargada de la gestión de información no estructurada que consiste en la creación de un sistema cliente-servidor capaz de insertar, buscar, actualizar y eliminar grandes cantidades de datos, contenido multimedia y datos de geolocalización sin una estructura fija. Se explicaran conceptos básicos como HTTP, REST, API, MVC, Node.js, Express.js, Bases de datos, SQL y NoSQL. Luego se mostrara como se aplicaron estos conceptos para la obtención de una arquitectura de software modular, flexible y altamente cohesionada que permite hacer uso de datos no estructurados

INTRODUCCIÓN

Hoy día las personas usan la tecnología para hacer de sus actividades diarias algo más sencillo y ordenado, dejan que la tecnología se encargue de sus tareas repetitivas, y la mayoría transforma estas facultades para suplir necesidades. En los últimos años la trasformación del software ha pasado de tipo tradicional a tipo social, lo cual ha tenido un gran impacto: el estilo de vida cotidiano ha cambiado y la inclusión de numerosas aplicaciones de uso diario han trasformado la manera de ver el mundo y mostrarse a él, es por ello que la evolución del software ha desencadenado en el desarrollo de nuevas tecnologías hechas para nuevas necesidades. Una perspectiva renovada hace posible la oportunidad de desarrollar desde la ingeniería multimedia nuevas tecnologías que aprovechan estos nuevos paradigmas.

Desde los años 60' se ha evidenciado el interés por el manejo ordenado de la información a gran escala; el uso del paradigma navigacional, bajo el estándar COBOL, el aporte de IBM con su propio sistema de bases de datos llamado IMS, el cual hacia uso de una jerarquía estricta para su navegación, los grandes aportes de Edgar Codd en una serie de artículos donde describe un nuevo enfoque para la construcción de bases de datos, que finalmente terminaron en las bases de datos relacionales, como se conocen hoy día, la inclusión de las hojas de cálculo como Lotus y el software de bases de datos dBASE, caracterizado en los 80`s porque era ligero y de fácil consulta para el usuario, hasta llegar a hacer uso del paradigma orientado objetos en las bases de datos, y empezar a hacer uso de lenguajes de consulta SQL.

En la actualidad se han creado las bases de datos de la nueva generación, son llamadas bases de datos NoSQL, las cuales están orientadas a datos no estructurados y ofrecen consultas más rápidas, escalabilidad horizontal. Su uso es más abierto ya que los proyectos se crean código abierto, hoy por hoy las más representativas son MongoDB, Memcached, Redis, CouchDB, Hazelcast, Apache Cassandra y Hbase.

El aporte de este tipo de bases de datos ha sido impactante y muchas de las aplicaciones más populares hacen uso de ellas por la multiplicidad de ventajas que ofrecen con el manejo de datos no estructurados, podemos encontrar actualmente varios ejemplos citados a continuación.

Las bases de datos no relacionales son el depósito que alimenta a MTV Networks, la próxima generación del CMS, que con el tiempo se utiliza para administrar y distribuir contenido para todos los principales sitios web de MTV Networks.

Codecademy, una página web dedicada a la enseñanza de programación., hace uso de de bases de datos no relacionales para almacenar todos sus cursos y datos necesarios para su funcionamiento.

Disney construyó un conjunto común de herramientas y APIs para todos los departamentos en el Grupo de Medios Interactivos, usando bases de datos no relacionales como un repositorio de objetos comunes para conservar información.

Las bases de datos no relacionales están siendo utilizadas por los juegos, alimentando los componentes, sirvieron para ea.com, rupture.com y el gestor de descargas de EA (Electronics Arts.

MoPub, CNN Turk, GitHub, Foursquare Viber, Grooveshark, el stack de LinkedIn entre muchos otros ya han comprobado las grandes ventajas y posibilidades que se proporciona con este tipo de almacenamiento.

A continuación se presenta un modelo de arquitectura de software portable, escalable y abierta que provee una solución para diversos tipos de problemas desde perspectivas generales hasta situaciones específicas, haciendo uso de software libre, bases de datos no relaciones, apropiando diferentes protocolos de comunicación, construyendo una herramienta para el manejo de grandes cantidades de información no estructurada que responde a las necesidades actuales en cuanto al manejo de la información.

En el capítulo 2 se explicara el problema del manejo de grandes cantidades de datos y porque cada día aumenta su complejidad, también se mostrara porque se propuso hacer uso de algunos conceptos como bases de datos no relaciones y uso de código no bloqueando para el desarrollo del servidor, luego se mostraran los objetivos generales y específicos planteados para desarrollar este proyecto y por último se podrá ver como se adaptó la metodología iterativa a la medida del proyecto. En el capítulo 3 se encuentra todo el soporte teórico del proyecto, aquí se explicaran conceptos como bases de datos, modelos de bases de datos, protocolos de información, arquitecturas de software y servidores web, además de introducir a tecnologías como MVC, SQL, NoSQL, HTTP, REST, Node.js y Express.js. Luego, en el Capítulo 4 se introducirá al problema del manejo de grandes cantidades de datos y la evolución de las redes sociales. Luego se determinan las características de uso, funcionales y no funcionales con la que debe cumplir la arquitectura y se formulan características individuales para el servidor y el cliente, por último se muestra como está planteada la arquitectura y como se manejaran los subsistemas de protocolo de información, modelo de la aplicación, y sistema de comunicación con la base de datos. Después en Capitulo 5 se plantea un ejemplo de aplicación, que consiste en una red social enfocada a la búsqueda de restaurantes. Allí se muestra como se adaptó la arquitectura propuesta en el capítulo 4 y se muestran los diseños realizados para la parte del cliente. En el capítulo 6 se muestran los satisfactorios resultados obtenidos, luego de hacer varias pruebas con el servidor, aquí se empiezan a evidenciar la eficiencia y velocidad de la arquitectura. Por ultimo en el capítulo 7 se formulan unas conclusiones de acuerdo a los resultados obtenidos, las ventajas de hacer uso de bases de datos no relacionales, protocolo de comunicación, hacer uso de código no bloqueante en la parte del servidor y por ultimo unas recomendaciones para el uso de estas tecnologías.

2. RESUMEN DE LA PROPUESTA

Desde la aparición de las aplicaciones sociales se ha hecho notable la necesidad de un sistema de almacenamiento flexible de alto rendimiento. En la actualidad la llegada de la web, el software como servicio, los servicios en la nube y las startups de éxito con millones de usuarios, han traído consigo problemas de alta de escalabilidad, el intento de los modelos de bases de datos relacionales por adaptarse en entornos difíciles como los anteriormente nombrados hacen que aumente la complejidad y sean menos intuitivos. Procesos en la forma de almacenar y consultar la información en las bases de datos relacionales hacen que el sistema sea poco eficiente y lo obliga a realizar almacenamiento de resultados en caches con el fin de evitar pesadas operaciones.

Para la solución del problema se ha propuesto trabajar sobre una base de datos no relacional, que se ocupa de datos de escala web, alta frecuencia de lecturas y escrituras, y no posee un esquema de datos rígido con el fin de asegurar respuestas rápidas del lado del servidor hacia el cliente permitiendo realizar consultas asíncronas en grandes magnitudes y asegurar una escalabilidad sin mayores limitaciones.

El uso de servidores con conexiones asíncronas permite mantener muchas conexiones abiertas y esperando, esto es una gran ventaja ya que evade la limitación en el número de conexiones y procesos que hagan uso de código bloqueante.

Se plantea una arquitectura dividida en cuatro capas que permite obtener un software altamente cohesionado, esto se hace con el fin de que la arquitectura pueda ser migrada sin mayores complicaciones, ya sea del lado del cliente o del servidor. Además permite realizar operaciones de lectura y escritura (I/O) de datos sin una estructura rígida con un bajo requerimiento de hardware y una rápida respuesta, teniendo en cuenta el elevado número de peticiones que puede llegar a tener simultáneamente.

2.1 OBJETIVOS

Objetivo General

Desarrollar una arquitectura de software escalable para la indexación, actualización y búsqueda de datos no estructurados mediante comunicación asíncrona cliente-servidor y capaz de alojar la información en una base de datos no relacional (No SQL).

Objetivos Específicos

- Diseñar una arquitectura para el almacenamiento de información.

- Diseñar un modelo para la aplicación encargado de recibir y procesar las peticiones del subsistema de comunicación entre el cliente y el servidor
- Implementar una arquitectura de software que genere un subsistema con un protocolo de comunicación y opere con el modelo del aplicativo.

- Desarrollar un cliente que brinde una interfaz de usuario y establezca una comunicación con el servidor.

- Desarrollar una aplicación sobre un dominio de problema específico con el fin de aplicar la arquitectura desarrollada.

2.2 METODOLOGIA

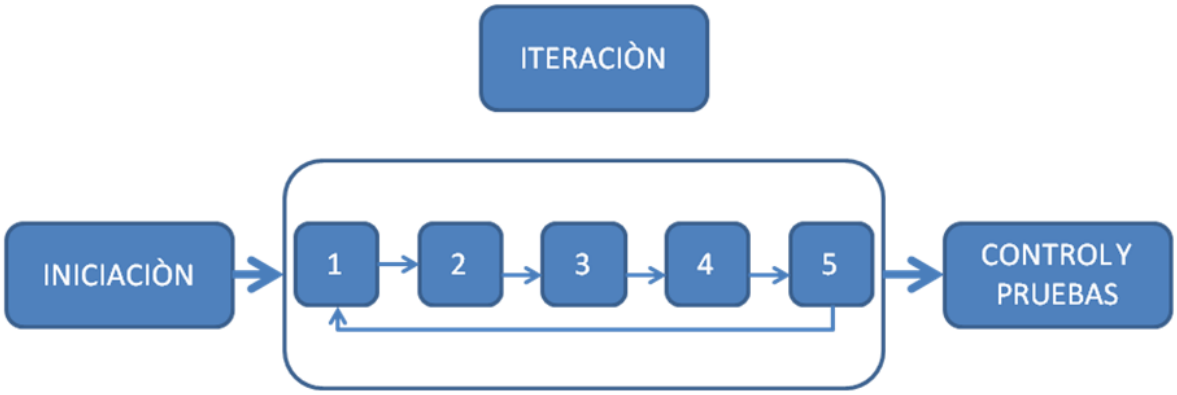


Figura2.1: Metodología e implementación, Diagrama de la metodología del proyecto.

1. Implementación de la interfaz de usuario y flujos de la aplicación.
2. Diseño e implementación de la capa de almacenamiento de información.
3. Diseño y desarrollo del modelo del aplicativo (API).
4. Diseño e implementación del protocolo de comunicación.
5. Desarrollo de los métodos en el cliente.

Figura 2.1: Metodología e implementación, Diagrama de la metodología del proyecto.

La metodología se basa en el método incremental iterativo, en la planeación se definieron los siguientes hitos:

- Consolidación del subsistema de indexación.
- Consolidación del subsistema de actualización.
- Consolidación del subsistema de eliminación.
- Consolidación del subsistema de búsquedas.

Estos hitos se componen de las siguientes etapas, obedeciendo a los objetivos específicos:

1. Diseño del flujo de la información:

   Consiste en definir como se comunicaran los subsistemas internamente, los protocolos que se emplearan, la manera de enviar y recibir las peticiones, datos y mensajes.

2. Diseño e implementación de la capa de almacenamiento de información:

   El diseño en la capa de almacenamiento se encarga de definir como se almacenaran los datos y como se relacionaran para asegurar que el manejo de la información y operaciones internas estén correctas y se facilite el manejo de la información.

3. Diseño y desarrollo del modelo del aplicativo (API):

   Se establece la estructura de la interfaz de programación y la forma en cómo podrá accederse a esta externamente. En el modelo, se implementan algunas reglas de negocio generalizando las acciones más comunes, validaciones, empaquetamiento de información y comunicación con el sistema de almacenamiento.

4. Diseño e implementación del protocolo de comunicación.

   Se define la forma de comunicarse desde el lado del cliente con el sistema, la arquitectura es accedida por medio de la API, el protocolo define la forma de enviar la información, la estructura de los mensajes, métodos para acceder a los recursos, y forma de direccionar correctamente las peticiones y datos.

5. Desarrollo de los métodos en el cliente que permita hacer pruebas.

   Los métodos en el cliente son la forma de consumir los servicios desarrollados en la arquitectura por medio del protocolo de comunicación, según el hito permitirá, indexar, actualizar, eliminar y consultar información de cualquier tipo, sin una estructura definida.

Los métodos en el cliente y la forma de presentar la información dependerán del dominio del problema específico o caso de ejemplo.

Pruebas: esta fase está diseñada para hacer las respectivas pruebas y de acuerdo a los resultados aplicar correcciones pertinentes para la optimización de la arquitectura.

Para asegurar que la arquitectura cumpla con factores de calidad y desempeño, se decidió hacer el desarrollo sobre estándares. Esto permite un óptimo uso de los recursos, hace que el sistema sea flexible y se adapte mejor a diferentes escenarios y acople mejor sus módulos internamente.

3. MARCO TEÓRICO

En esta sección se hará una breve introducción a los conceptos que sustentan la arquitectura propuesta en este proyecto, describiendo los componentes más relevantes en la construcción de sistemas similares y referenciando estándares establecidos que serán pilares en el desarrollo, exponiendo los modelos y herramientas tanto tradicionales así como las nuevas prácticas.

3.1 GESTIÓN DE LA INFORMACIÓN

Un modelo de datos es aquel que se encarga de gestionar la información determinada, la gestión de la información es un modelo que construye un lenguaje que sirve para comunicarse con la base de datos genérica, definiendo el tipo de base que se debe emplear para que esta sea coherente con la realidad y la información que será tratada. Además, debe mostrar implícitamente los procesos de inserción, borrado, actualización y consulta en su misma interacción.

Los modelos pueden tener estructuras de datos definidas o no y presentar restricciones de integridad es decir, el conjunto de condiciones que deben cumplir los datos para reflejar correctamente la realidad. A este tipo de datos se les conoce en dos categorías: estructurados y no estructurados. Los datos estructurados son aquellos que poseen una estructura fija, estos fueron una solución a problemas de abstracción y manejo de grandes cantidades de información por muchos años pero últimamente, están quedando encasillados en el manejo de datos de tipo bancario y corporativo, esto sucede si la estructura es lo suficientemente rígida para acomodarse a los modelos. La información que se maneja en el mundo actual usualmente no posee estructura fija, los datos multimedia requieren de un manejo más libre y un almacenamiento hecho a la medida que le permita al usuario y al desarrollador hacer uso de una tecnología mucho más flexible.

En cuanto a los datos no estructurados (o información no estructurada), hacen referencia a la información que no tiene un modelo de datos predefinidos. La información no estructurada son textos que representan datos tales como fechas, números, imágenes, videos y en general, contenido multimedia. Al utilizar este tipo de datos se pueden presentar irregularidades y ambigüedades que dificultan el entendimiento de la metodología de los programas informáticos tradicionales, con respecto a los datos almacenados y la forma de estructurar las bases o anotaciones (semánticamente marcado) en los documentos.

En 1998, Merrill Lynch citó una regla de oro: "*Alrededor del 80-90% de toda la información empresarial potencialmente utilizable puede proceder en forma no estructurada*".

A continuación, se especificarán los tipos de bases de datos que son apropiados para el manejo de la información estructura y no estructurada.

3.1.1 Bases de Datos

Una base de datos es una colección organizada de datos que sirve para modelar los aspectos relevantes de la realidad de una manera que apoye los procesos que requieren de la gestión de la información.

Los sistemas de gestión de bases de datos (DBMS) son aplicaciones que pueden interactuar con el usuario, con otras aplicaciones y/o con ella misma, para capturar y analizar datos previamente diseñados. Existen dos tipos de DBMS: Los SQL (Ver apartado 3.1.4) y las NoSQL (Ver apartado 3.1.5), que tienen un propósito general que permiten la definición, creación, consulta y actualización de una bases de datos. Los DBMS más conocidos son MySQL , PostgreSQL , SQLite , Microsoft SQL Server , Microsoft Access , Oracle , Sybase , dBASE , FoxPro y IBM DB2 . Una base de datos no es generalmente portable a través de diferentes DBMS, pero diferentes DBMS pueden interpolar utilizando estándares como SQL y ODBC o JDBC para permitir a una aplicación única trabajar con más de una base de datos [1].

3.1.2 Dbms de Uso General y Especial

Un DBMS se ha convertido en un sistema de software complejo y su desarrollo suele requerir gran cantidad de personal humano que trabaja durante varios años en la elaboración y mejoramiento del mismo [2]. Algunos DBMS de propósito general como Adabas, Oracle y DB2 han sido sometidos a mejoras desde 1970.

Los DBMS de propósito general tienen como objetivo satisfacer las necesidades de las aplicaciones hasta donde les sea posible. Entre más específico, más complejo. Sin embargo, el hecho de que el costo de desarrollo es considerable y puede ser distribuido entre un gran número de usuarios, significa que a menudo son el enfoque más rentable, pero un DBMS de propósito general no es siempre la solución óptima.

En algunos casos un DBM de propósito general puede introducir una sobrecarga innecesaria, por ello existen DBMS
de propósito especial.

DBMS de propósito especial: Para definirlo, se hará uso de un ejemplo común como es el sistema de dirección de correo electrónico: Los sistemas de correo electrónico están diseñados para optimizar el manejo de los mensajes de y no necesitan una parte significativa de la funcionalidad de propósito general DBMS.

Muchos DBMS tienen aplicaciones que poseen acceso a la base de datos en nombre de los usuarios finales, sin exponer la interfaz directamente. Los programadores de aplicaciones pueden usar un protocolo de conexión directa o una interfaz de programación de aplicaciones (API). Diseñadores y administradores de bases de datos interactúan con el DBMS a través de interfaces dedicadas a construir y mantener las aplicaciones de la base de datos, por lo que necesitan mayor conocimiento y comprensión del funcionamiento de los DBMS, las interfaces externas y los parámetros de ajuste.

Las bases de datos de uso general suelen ser desarrolladas por una organización o comunidad de programadores, mientras que otro grupo construye las aplicaciones que las utilizan. En muchas empresas existen administradores especializados en bases de datos que las mantienen, generan informes y pueden trabajar en el código que se ejecuta en las mismas, sin hacer uso del software especializado.

3.1.3 Modelo de Bases de Datos

Un modelo de base de datos determina la estructura lógica de una base de datos y precisa de qué manera los datos se pueden almacenar, organizar, y manipular. El ejemplo más claro de un modelo de base de datos es el modelo relacional (o la aproximación de SQL relacional), que usa un formato basado en tablas.

Modelos comunes de datos lógicos para bases de datos:

- Modelo de base de datos jerárquica
- Modelo de red
- Modelo relacional
- Modelo de entidad-relación
- Modelo de entidad-relación mejorada
- Modelo de objetos
- Modelo de Documento
- Modelo de entidad-atributo-valor
- Estrella esquema

Otros modelos incluyen:

- Modelo asociativo
- Modelo multidimensional
- M odelo multivalor
- Modelo semántico
- Base de datos XML
- Gráfico Nombrado

3.1.4 SQL

Originalmente basado en el álgebra relacional y el cálculo relacional de tuplas, SQL consiste en un lenguaje de definición y manipulación de datos. El ámbito de aplicación de SQL incluye inserción de datos, consulta, actualización y eliminación, esquema de creación y modificación, y el control de acceso a datos. Aunque SQL se describe a menudo, y en gran medida como un lenguaje declarativo (4GL), también incluye elementos procesales. SQL fue uno de los primeros lenguajes comerciales para el modelo relacional de Edgar F. Codd, como se describe en su influyente papel en 1970 "Un modelo relacional de datos para grandes bancos de datos compartidos" [3]. A pesar de que no cumpla en su totalidad con el modelo relacional, como es descrito por Codd, se convirtió en el lenguaje de base de datos más utilizado [4] [5].

SQL se transformó en un estándar del American National Standards Institute (ANSI) En 1986, y de la Organización Internacional de Normalización (ISO) en 1987 [6]. Desde entonces, el estándar se ha mejorado varias veces con características adicionales. Pese a esto, el código no es completamente portable entre diferentes sistemas de bases de datos y puede conducir a la dependencia de un proveedor.

3.1.5 No SQL

Es un subconjunto de bases de datos que difiere en varios modos de bases de datos tradicionales (RDBMS), no tiene esquema, no permite JOINS (La sentencia JOIN en SQL permite combinar registros de dos o más tablas en una base de datos relacional), no intentan garantizar ACID (Protocolo que asegura la transacción de información) y escalan horizontalmente. De hecho, las bases de datos NoSQL como las SQL poseen un tipo de almacenamiento estructurado [7]. Esto quiere decir que manejan datos no estructurados que en definitiva, son guardados de manera estructurada en el disco de almacenamiento.

El término NoSQL fue acuñado en 1998 por Carli Strozzi y resucitado en 2009 por Eric Evans durante un evento organizado por Johan Oskarsson en el que se hablaba de las crecientes bases de datos que no garantizaban el uso del ACID.

La arquitectura menudo ofrece sólo garantías de consistencia débiles, como por ejemplo consistencia eventual o transacciones restringidas a elementos de datos simples. Emplean una arquitectura distribuida donde los datos se guardan de modo redundante en distintos servidores a menudo usando tablas hash distribuidas. Por lo general, se ofrecen estructuras de datos sencillas como arreglos asociativos o almacenes de pares clave-valor

La taxonomía de estas bases, de acuerdo a su implementación es:

- Almacenes de clave-valor: Se pone una palabra clave para identificar un campo específico, el cual contiene un valor de tipo genérico.

- Almacenes de familia de columnas: Son valores del mismo tipo que representan una clave.
- Almacenes de documentos: a partir de la agrupación de varias clave-valor se construye un documento que será almacenado en la base de datos.
- Almacenes de grafos: Un documento puede estar conectado con varios documentos a la vez.

3.2 ARQUITECTURA CLIENTE - SERVIDOR

Una arquitectura cliente servidor posee un flujo definido en sus procesos y su funcionamiento es básicamente el despliegue exitoso de este: los clientes inician solicitudes a los servidores, estos se encargan de tramitar las solicitudes y devolver las respuestas apropiadas. El cliente comienza a enviar solicitudes cuando está listo para hacer la transición a un nuevo estado. Mientras que una o más solicitudes son excepcionales, el cliente se considera en transición, este estado en arquitecturas como REST no existe y será explicado con mayor detalle en el capítulo 3.3.2. La representación de cada estado de la aplicación contiene enlaces que pueden usarse la próxima vez que el cliente decide iniciar un nuevo estado de transición [20].

Las solicitudes y las respuestas se construyen alrededor de la transferencia de las representaciones de los recursos. Un recurso puede ser cualquier concepto coherente y significativo que se podrá utilizar dentro de la arquitectura. Una representación de un recurso suele ser un documento que recoge el estado actual o previsto del mismo.

Arquitecturas cliente servidor convencionalmente hacen uso de estándares de estilo REST (Ver capítulo 3.3.2) y poseen un protocolo de comunicación definido.

3.3 PROTOCOLOS DE COMUNICACIÓN

En informática y telecomunicación, un protocolo de comunicaciones es un conjunto de reglas y normas que permiten que dos o más entidades de un sistema de comunicación se comuniquen entre ellos, para transmitir información por medio de cualquier tipo de variación de una magnitud física. Se trata de las reglas o el estándar que define la sintaxis, semántica y sincronización de la comunicación, así como posibles métodos de recuperación de errores. Los protocolos pueden ser implementados por hardware, software, o una combinación de ambos [17], en entornos web tradicionalmente han sido desarrollados protocolos a la medida para la comunicación y manipulación de recursos en internet.

3.3.1 HTTP

Protocolo de transferencia de hipertexto (HTTP) es un protocolo de aplicación de los sistemas de información hipermedia distribuidos y colaborativos [18]. HTTP es la base de

la comunicación de datos para la World Wide Web y ha encajado a la perfección en arquitecturas de tipo REST.

El hipertexto es texto estructurado que utiliza enlaces lógicos (hipervínculos) entre los nodos que contienen texto. El desarrollo de estándares de HTTP ha sido coordinado por el Grupo de Trabajo de Ingeniería de Internet (IETF) y el World Wide Web Consortium (W3C).

3.3.2 REST

Transferencia de estado representacional (REST), es un estilo de arquitectura de software para sistemas distribuidos como World Wide Web. REST se ha convertido en un predominante modelo de diseño en web API.

REST afirma que la web ha disfrutado de escalabilidad como resultado de una serie de diseños fundamentales identificados como:

- Un protocolo cliente/servidor sin estado: cada mensaje HTTP contiene toda la información necesaria para comprender la petición. Como resultado, ni el cliente ni el servidor necesitan recordar ningún estado de las comunicaciones entre mensajes.

- Un conjunto de operaciones bien definidas que se aplican a todos los recursos de información: HTTP en sí define un conjunto pequeño de operaciones, las más importantes son POST (Almacenar), GET (Obtener), PUT (Actualizar) y DELETE (Borrar). Cuando una aplicación hace uso de estos cuatro métodos, se dice que es una aplicación RESTful.

- Una sintaxis universal para identificar los recursos. En un sistema REST, cada recurso es direccionable únicamente a través de su URI.

- El uso de hipermedios, tanto para la información de la aplicación como para las transiciones de estado de la aplicación: la representación de este estado en un sistema REST son típicamente HTML o XML. Como resultado de esto, es posible navegar de un recurso REST a muchos otros, simplemente siguiendo enlaces sin requerir el uso de registros u otra infraestructura adicional.

3.3.2.1 API REST

Una API o interfaz de programación de aplicaciones, es algo así como un contrato de codificación: especifica las formas en que un programa puede interactuar con una aplicación. Es una colección de recursos con tres aspectos definidos:

- El URI base para la API Web, como ***http://ejemplo.com/recusos/***
- El tipo de datos soportados por la API web. Esto es a menudo JSON, pero puede ser cualquier otro tipo de medio de Internet válido siempre que se trate de un estándar de hipertexto válido.

- El conjunto de las operaciones de apoyo de la API web utilizando métodos HTTP (por ejemplo, GET, PUT, POST o DELETE) [19].

3.4 ARQUITECTURAS DE SOFTWARE

La arquitectura de software es en esencia el proceso de definición de una solución estructurada que cumple con todos los requisitos técnicos y operativos, encargada de optimizar los procesos comunes de calidad tales como el rendimiento, la seguridad y capacidad de administración.

Philippe Kruchten, Grady Booch, Kurt Bittner y Rich Reitman, inspirados en la postura de Mary Shaw y David Garlan (Shaw y Garlan 1996), han definido la arquitectura de software como:

"La Arquitectura de software abarca el conjunto de decisiones significativas acerca de la organización de un sistema de software, incluyendo la selección de los elementos estructurales y sus interfaces con las cuales el sistema se compone; el comportamiento específica la colaboración entre los elementos; la composición de estos elementos estructurales y de comportamiento en subsistemas más grandes, y un estilo arquitectónico que guía esta organización. La Arquitectura de software también incluye la funcionalidad, facilidad de uso, flexibilidad, rendimiento, reutilización, comprensibilidad, las limitaciones económicas y la tecnología, ventajas, desventajas y las preocupaciones estéticas".

Los sistemas deben ser diseñados teniendo en cuenta el usuario, el sistema (la infraestructura de las Tecnología de la Información) y los objetivos del problema propuesto para desembocar en una lógica de negocio bien construida. Para cada una de estas áreas se deben esbozar escenarios clave e identificar los atributos de calidad importantes como fiabilidad y escalabilidad.

3.4.1 Modelo Vista Controlador - MVC

Modelo-Vista-Controlador (MVC) es un patrón de arquitectura de software que separa la representación de la información de la interacción con el usuario [21] [22]. El modelo consta de los siguientes elementos: datos de la aplicación, reglas de negocio, la lógica del negocio y funciones a desplegar. Una vista puede ser identificada como cualquier representación de salida de datos, es decir, gráficos o diagramas. Es posible generar u obtener múltiples vistas de los mismos datos, como un gráfico de barras para la gestión y la vista en tablas para los contadores. El controlador es básicamente es disparador o interprete de entradas al sistema que traduce las intenciones convirtiéndolas en comandos para el modelo o la vista [23].

3.5 SERVIDOR WEB

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

Para la transmisión de todos estos datos suele utilizarse un protocolo HTTP, perteneciente a la capa de aplicación del estándar OSI (Open System Interconection). El término HTTP también se emplea para referirse al ordenador que ejecuta el programa [10].

3.5.1 Node.Js

Node.js es un entorno de programación en la capa del servidor basado en el lenguaje de programación Javascript, con I/O de datos en una arquitectura orientada a eventos y basado en el motor Javascript V8. Fue creado el enfoque de ser útil en la implementación de programas de red altamente escalables, como por ejemplo, servidores web [12]. Fue creado por Ryan Dahl en 2009, y su evolución está apadrinada por la empresa Joyent [13] [14].

Node.js es similar en su propósito a Twisted de Python, Perl Object Environment para Perl, libevent para C y EventMachine para Ruby. Al contrario de la mayoría del código JavaScript, no se ejecuta en un navegador, sino en el lado del servidor [15]. Node.js incluye un entorno REPL(Read Eval Print Loop) para depuración interactiva y brinda la posibilidad de hacer uso de frameworks para tener control sobre las peticiones y respuesta de los recursos a los que se acceden en el servidor.

3.5.1.1 Express.Js

Express es un framework de aplicaciones web para Node.js flexible que proporciona un robusto conjunto de características para crear aplicaciones web de una o varias páginas, e híbridos [16], direccionando las peticiones hechas del lado del cliente de manera correcta y obteniendo un reporte de las transacciones y procesos en el servidor.

4. ARQUITECTURA DE SOFTWARE

En este capítulo, se dan la especificaciones que debe cumplir la arquitectura de software de cara a los objetivos planteados en el proyecto, se identificarán sus requerimientos, el dominio general del problema, las características de uso, su público objetivo, el diseño, el almacenamiento de información y el modelo de la arquitectura.

4.1 REQUERIMIENTOS DE LA ARQUITECTURA

En este parágrafo, se muestran los requerimientos para el despliegue de la arquitectura que fue identificado luego de realizar el análisis y depuración de la información del capítulo 3, donde se han recopilado las formalidades y teorías que soportan el proyecto.

Con el objetivo de evidenciar el funcionamiento de la arquitectura de software implementada, se ha definido un dominio del problema que contribuyó para establecer los múltiples requerimientos del sistema, abarcando aspectos de hardware y software.

4.1.1 Dominio del Problema

Actualmente es fácil identificar que la tecnología cambio radicalmente la forma del comportamiento humano, la mayoría de las actividades diarias en diferentes ámbitos profesionales y sociales están ligadas a la tecnología, la masificación de la redes sociales hacen que las personas interactúen constantemente y compartan información, esta información es el foco principal del problema, en redes sociales las personas interactúan con diferentes elementos multimedia subiendo y descargando grandes cantidades de información que les permite mostrar diferentes actividades, facetas o localizaciones en las que han estado; en pocas palabras, socializar sus actividades virtualmente.

El tema de manipular la información trasciende a otros campos de la vida como la educación, la salud, el mercadeo, el entretenimiento por nombrar algunos. La cantidad de información que se maneja es bastante grande y el problema radica en que la mayoría de esta información no puede ser accedida fácilmente, el volumen de almacenamiento hace que los sistemas se carguen y su funcionamiento no sea ideal, de aquí nace la idea de crear una arquitectura que esté en capacidad de recolectar grandes cantidades de elementos multimedia que luego puedan ser accedidos de manera ágil y eficiente, brindando la posibilidad de crear nuevas experiencias para los potenciales usuarios en diferentes ámbitos de la vida.

4.1.1.1 Análisis del dominio

Técnicamente hablando el problema se sintetiza en el manejo de grandes cantidades de información y objetos multimedia, el objetivo es que la información pueda ser escrita,

consultada, eliminada y actualizada de una manera fácil, ágil y exista un registro confiable de los datos que se almacenan.

Para garantizar todos los requerimientos nuestra arquitectura estará diseñada modularmente, separamos el almacenamiento, la comunicación, la interfaz de programación y el cliente. Las ventajas de esto es que podemos modificar cualquiera de los módulos que la componen sin cambiar por completo los otros, es decir que si se quiere el tipo de almacenamiento puede variar entre una base de datos a otra, el protocolo de comunicación podría cambiarse en caso de ser necesario, y la arquitectura puede ser usada desde cualquier dispositivo para que haya sido creado un cliente, la versatilidad nos permite desarrollar software para clientes móviles, de escritorio, tabletas, o cualquier dispositivo electrónico que posea conexión a internet y su software pueda ser manipulado, todo esto con el fin de construir una herramienta que permita desarrollar sistemas flexibles pero robustos, sin importar la plataforma, que se adapten a una gran cantidad de problemas o situaciones donde se manipule información.

A continuación son descritas las características principales de la arquitectura.

4.1.1.2 Características de Uso

- El usuario empleará la aplicación para acceder a los objetos multimedia mediante búsquedas por texto.
- El usuario podrá compartir objetos multimedia con otros usuarios, y de esta manera alimentar la base de datos.

4.1.1.3 Características Funcionales

- Compartir localización de los objetos multimedia.
- Tomar fotos del lugar, servicio o producto.
- Ingresar información, descripción o datos de los elementos.
- Realizar búsquedas de los elementos por texto.
- Realizar búsquedas de los elementos por locación.
- Crear usuario.
- Iniciar sesión Usuario.
- Actualizar información.
- Eliminar información.

4.1.1.4 Características No Funcionales

- Acceso a internet.
- Servidor de rápida respuesta y activo las 24 horas del día.

4.1.1.5 Público Objetivo

Todas las personas a las cuales les competan las necesidades nombradas en el apartado 3.1.1, sin importar la edad, sexo u otro tipo de característica.

4.1.2 Requerimientos del Cliente

En este parágrafo se mostraran las principales características propuestas para un cliente que desee hacer uso de la arquitectura desarrollada

4.1.2.1 Características Técnicas

- Dispositivos con conexión a Internet
- Dispositivo capaz de hacer uso de contenido multimedia
- Dispositivos con la posibilidad de usar tecnologías web que permitan usar protocolos de comunicación ya sea de manera nativa, o con el uso de Frameworks.
- Poseedor de GPS o sistemas de localización.

4.1.3 Requerimientos del Servidor

Aquí se nombraran todos los requisitos necesarios que debe cumplir el servidor para que la arquitectura pueda ser implementada.

4.1.3.1 Requerimientos funcionales

- Capaz de alojar diferentes lenguajes de servidor
- Escalable horizontalmente de manera rápida.
- Calificado para el alojamiento de diferentes tipos de bases de datos.
- Capaz de interpretar lenguajes dinámicos, débilmente tipados.
- Capaz de alojar bases de datos no relacionales.

4.1.3.2 Requerimientos No funcionales

- Apto para versionar cada despliegue público del servicio.
- Idóneo para monitorear el flujo de información.
- Amplio en documentación y soporte.

4.1.3.3 Características Técnicas

- Servidores capaces de resolver múltiples peticiones utilizando un entorno de programación flexible y bases de datos No Relacionales.

Para cumplir con los requerimientos definidos es necesario desarrollar un sistema modular para hacer la arquitectura más flexible y no dependiente de tecnologías o herramientas específicas, dividendo la administración de los datos, la comunicación, el almacenamiento y la presentación.

Es importante hacer uso de bases de datos no relacionales y escalables horizontalmente para almacenar la información. Construir un servidor que permita desplegar código escrito en un lenguaje débilmente tipado, un cliente que pueda desplegar información multimedia y comunicarse mediante protocolos web ya sea de forma nativa o con el uso de frameworks, es vital que el dispositivo que haga uso del sistema cuente con una conexión a internet.

4.2 DISEÑO DE LA ARQUITECTURA

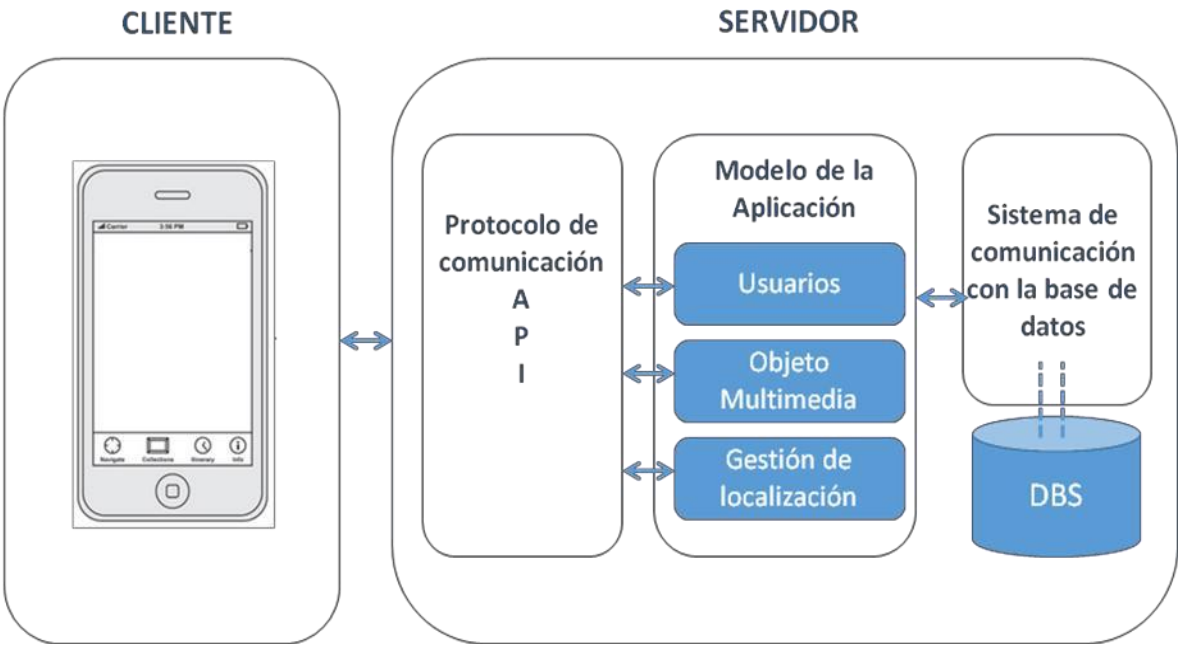


Figura 4.1: Arquitectura Cliente/Servidor, Diagrama General de la arquitectura.

De acuerdo a los requerimientos planteados en 4.1 se formula una arquitectura cliente/servidor dividida en cuatro capas. Estas se basan en módulos altamente cohesionados y de bajo acoplamiento que permiten obtener un software compatible, la distribución de estos módulos se evidencia en la figura 4.2 dividido modularmente:

1. Cliente: Consume la arquitectura mediante la API provista, envía y recibe peticiones y presenta las respuestas en un dispositivo de entrada y salida.
2. Protocolo de comunicación: Es el encargado de recibir las peticiones del cliente y enviarlos al modelo de la aplicación.

3. Modelo de la aplicación: Es el encargado de procesar las peticiones que llegan del protocolo de comunicación y enviar operaciones al Sistema de comunicación con la base de datos.
4. Sistema de comunicación con la base de datos: Es el encargado de insertar, modificar, borrar y obtener información de la base de datos.

El propósito de construir una arquitectura modular tiene como objetivo: asegurar que el software pueda ser migrado a diferentes plataformas sin mayores complicaciones, ya sea del lado del cliente o del servidor. Estos principios favorecen los objetivos de calidad de la arquitectura.

La arquitectura permite realizar operaciones de lectura y escritura (I/O) de datos sin una estructura rígida, el formato de los datos es de tipo JSON, con un bajo requerimiento de hardware y una rápida respuesta, teniendo en cuenta el elevado número de peticiones que puede llegar a tener simultáneamente.

El diseño de la base de datos, es no relacional, usando Mongo DB, lo cual permite ocuparse de datos a escala web, Node js para el entorno de programación ya que nos permite hacer conexiones asíncronas debido a su capacidad de mantener muchas conexiones abiertas en espera, esto es una gran ventaja ya que se evade la limitación en el número de conexiones y necesidad de software externo, que haga uso de un código bloqueante. Si en nuestra arquitectura existe una operación bloqueante (I/O por ejemplo), algunos de estos servidores crearán entonces otro hilo en segundo plano para responder al proceso, pero no lo harán sistemáticamente por cada conexión.

4.2.1 Arquitectura De Almacenamiento De Información

En este parágrafo se explicará cómo se hizo uso de las tecnologías con las que se implementó el módulo de "Sistema de comunicación con la base de datos" visto en la figura 4.1.

4.2.1.1 Base de Datos

Teniendo en cuenta los tipos de bases de datos vistas en el capítulo 3.1, encontramos a Mongo DB, una base de datos NoSQL que nos brinda la posibilidad de almacenar la información como la arquitectura lo requiere. Para ello, se ha usado un modelo de almacenamiento basado en colecciones de documentos, donde se crean diferentes entidades para dividir la información almacenada como se muestra en la Figura 4.2.

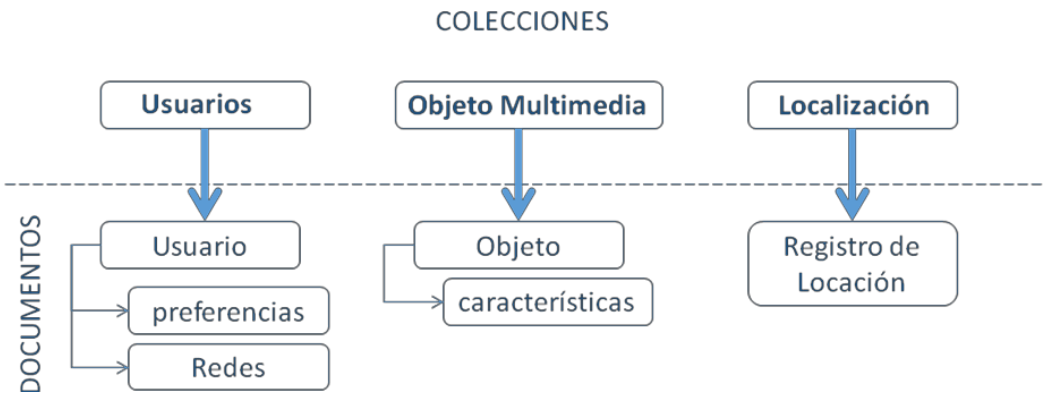


Figura 4.2: Estructura de almacenamiento, colecciones internas en la base de datos.

Las colecciones están identificadas como usuarios, objeto multimedia y gestión de la localización, cada una propietaria de un modelo definido y en términos de almacenamiento una colección en la base de datos con múltiples documentos. Dicho modelo será explicado con mayor detalle en el modelado de datos.

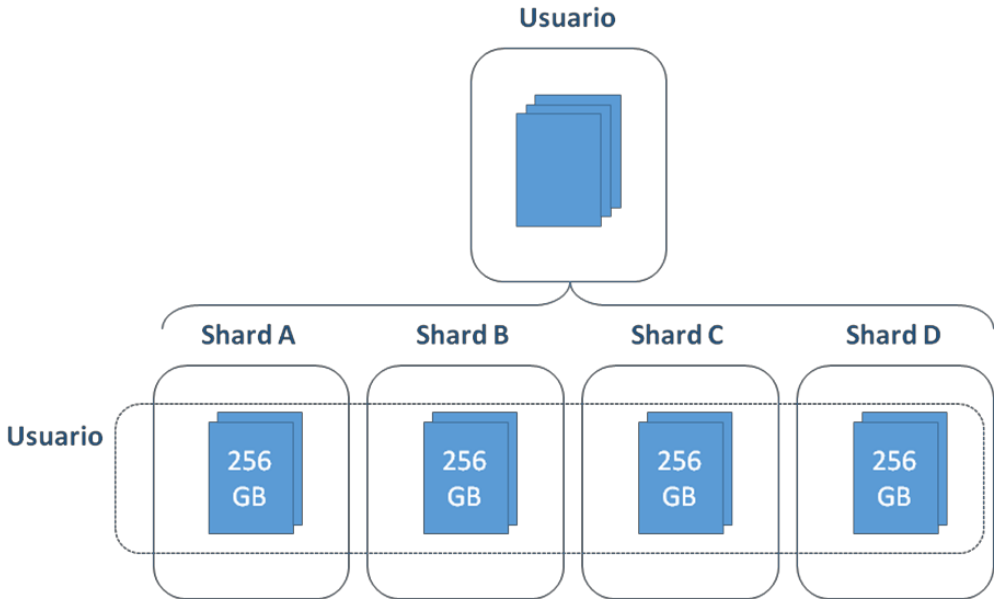


Figura 4.3: Estructura de almacenamiento Interno de las colecciones en Mongo DB.

El almacenamiento interno en Mongo DB de las colecciones se hace mediante “Sharding”, como se muestra en la figura 4.3, y consiste en dividir el conjunto de datos y distribuirlos a través de múltiples servidores, o fragmentos, evitando el estrés en el servidor y evadiendo problemas como el agotamiento de CPU y carga en la RAM.

Mongo DB permite hacer un uso híbrido entre las bases de datos no relacionales y las relacionales, como se explicó anteriormente en el capítulo 3.1.5. Es por ello que las entidades propuestas se interrelacionan entre sí para aprovechar al máximo las ventajas de ambos modelos de bases de datos para el almacenamiento, haciendo uso de los diferentes

tipos de consultas que proporciona las bases de datos de este tipo, se optimizan las búsquedas y se generan respuestas más rápidas para solucionar algunos problemas complejos que se habían considerado en el inicio del proyecto, tales como búsqueda por geolocalización o búsquedas por regresiones.

4.2.1.2 Sistema de Comunicación con la Base de Datos

Este sistema se encarga de comunicarse con la base de datos realizando las operaciones básicas (insertar, modificar, indexar y eliminar) sobre los datos procesados en el modelo. D icha comunicación se genera con el objetivo de construir un sistema transparente para la capa del modelo de la arquitectura, al realizar operaciones con la base de datos.

La función primaria de esta capa es crear el flujo entre la base de datos y el modelo de la arquitectura para que esta quede altamente cohesionada. Se genera una capa intermedia entre el modelo del aplicativo y la base de datos para solventar migraciones futuras de la información, en caso de necesitar hacer cambio a otro tipo de base de datos. Esto evita modificar todo el código en la capa del modelo de la aplicación.

La capa posee dos tipos de métodos: En primer lugar están los privados, que son generales y se encargan de crear, indexar, modificar y eliminar documentos o colecciones. El segundo método hace referencia a los públicos; estos consumen de los métodos privados, pero difieren en que son específicos para el dominio del problema. En este tipo de métodos (públicos) verifican los datos de acuerdo a las estructuras creadas a la medida del cliente.

El modelo de almacenamiento diseñado para la base de datos: consiste en crear una colección para almacenar documentos, compuestos de campos, los cuales también pueden contener documentos o colecciones.

Para crear una conexión con la base de datos se deben seguir los siguientes pasos:

1. Se debe crear una conexión con el servidor (La base de datos es un servidor aparte), por ello se debe especificar la URL donde está alojada y el puerto por el que está recibiendo peticiones (host y port respectivamente). También se debe manifestar el nombre de la base de datos (dbName) y las opciones de conexión (options).

```
db = new Db(dbName, new Server(host, port, options));
```

2. Luego de crear una conexión con la base de datos, esta debe abrirse (En caso de no existir una base de datos nombrada de la forma especificada anteriormente Mongo DB creará una nueva). Aquí se especificará una función que se ejecutará cuando el proceso finalice. Esta función se denomina “callback”, que hace referencia a una buena práctica para evadir problemas de código bloqueante. En ella, se deben especificar acciones que se hagan luego de conectarse exitosamente con la base de datos o respuestas en caso de no establecer una conexión.

```
db.open(callback)
```

3. Dentro del "callback" de la función "open", se debe verificar que la conexión fue exitosa. Luego se debe autenticar para asegurar que nadie más puede modificar la base de datos. Se debe especificar el nombre del usuario, la contraseña y la función con las acciones que se realizarán cuando el proceso de autenticación finalice.

```
db.authenticate(username, password, callback)
```

4. En la función "Callback" debe verificarse si la autenticación fue exitosa, de ser así, se crean las colecciones correspondientes, especificando el nombre de la colección y las acciones a realizar luego de su creación. Las colecciones se crean de la siguiente manera:

```
db.createCollection(collectionName, callback)
```

Cabe destacar que para la creación de colecciones, MongoDB sigue las mismas reglas por las que se rige la creación y apertura de la base de datos, es decir, que si dicha colección no se ha creado en la base de datos se creará una nueva.

4.2.2 Diseño del Flujo de la Información

Este parágrafo hace referencia del diseño que se planteó para el flujo de información en la arquitectura presentada en la figura 4.2. Para esta, se han contemplado diferentes aspectos, los cuales se encargan de definir el tipo de sistema de comunicación, haciendo uso de una arquitectura cliente-servidor y permitiendo eliminar el estado en las peticiones.

En la figura 3.2, se presenta la estructura del flujo de la información a manera general, donde existen dos sistemas que se comunican entre sí: El cliente encargado de enviar peticiones al servidor, de acuerdo con las necesidades del usuario final y el servidor, encargado de recibir y procesar las peticiones generadas por el cliente, almacenando la información contenida en dichas peticiones para posteriormente, enviar una respuesta al cliente.

La figura 3.2 presenta la estructura interna de comunicación del servidor, la cual consta de tres capas que se comunican secuencialmente y de manera bidireccional.

La primera capa es el protocolo de comunicación, encargada de operar con el modelo de la arquitectura; La segunda capa, llamada el modelo de la arquitectura, se ocupa de procesar las peticiones de la capa de protocolo de comunicación y enviar los datos a la capa de almacenamiento de información, que se encarga del procesamiento de la información contenida en la base de datos.

El diseño del flujo de la información consiste en establecer los actores encargados para procesar la información que entra al sistema y así mismo, establecer el flujo de éxito en las diferentes capas de la arquitectura.

El flujo exitoso en una petición, indistintamente de que tipo sea y procesalmente hablando, se manifiesta cuando:

1. La petición es enviada por el cliente mediante el protocolo de comunicación.
2. El protocolo envía la petición al modelo del aplicativo.
3. La petición es recibida y procesada en el modelo del aplicativo.
4. La capa de modelo del aplicativo envía la información a la capa de almacenamiento.
5. La capa de almacenamiento recibe y procesa la información.
6. La capa de almacenamiento envía una respuesta al modelo.
7. El modelo procesa la información de la respuesta de la capa de almacenamiento.
8. El modelo envía una respuesta al cliente mediante el protocolo de información.

4.2.3 Modelo de la Aplicación

Para la gestión de la información, referente a consultas y actualizaciones, se ha definido un modelo llamado "Modelo de la aplicación" (Como se observa en la figura 4.2), basado en la recepción de una petición y el envío de una respuesta al haber finalizado el proceso en el servidor. El Framework Express.js, explicado en el capítulo 3.5.1.1, hace posible desglosar las peticiones en el servidor y da una estructura genérica en la transferencia de información.

El modelo está compuesto por tres módulos encargados de la lógica del negocio con respecto a los usuarios, los objetos multimedia y la gestión de la localización en cada lugar.

Cada módulo se encarga de recibir las peticiones del cliente y realizar diferentes operaciones a los paquetes de información. El principal objetivo del modelo es brindar una comunicación transparente entre el cliente y la base de datos y así mismo, brindar una interfaz de programación de la arquitectura.

Internamente, los módulos realizan procesos de encripción de datos y eliminan al máximo la interpretación de datos para enviar paquetes de información claros a la capa de almacenamiento, siempre están abiertos a las peticiones del cliente, mediante el protocolo de comunicación establecido, y a las respuestas emitidas por las capas de almacenamiento, actuando como un moderador en el flujo de la información.

El desarrollo de cada módulo permite acceder a diferentes métodos públicos expuestos al cliente y su funcionamiento se construye en gran parte sobre métodos privados que se encargan de hacer labores específicas según el dominio.

4.2.4 Diseño del Protocolo de Comunicación Sistema

El diseño del protocolo de comunicación, como se observa en la figura 4.1, se encarga de transmitir los mensajes de una manera simple y concisa hacia el modelo de la aplicación.

Generando un sistema RESTful (termino hace referencia a un sistema que hace uso de todas las operaciones de REST, capítulo 3.3.2) para proporcionar una conexión eficiente entre el modelo y el cliente, actuando a manera de controlador y brindando una API, haciendo uso de un contrato de codificación para acceder a los recursos construidos.

Estos recursos son accedidos mediante un mensaje HTTP el cual contiene toda la información necesaria para comprender una petición y realizar una acción, haciendo uso de una sintaxis universal para identificarlos. Cada recurso es direccionable únicamente a través de su URI (Identificador Uniforme de Recursos).

Para acceder a un recurso definido en el modelo, la estructura correcta ha de ser la siguiente:

http://findmyfood.jit.su/checkIn/createcheckIn/username

Para establecer el protocolo de comunicación por medio de la estructura mencionada, se ha desarrollado un conjunto de operaciones de apoyo de la API utilizando métodos http que están estructurados de la misma forma. Allí se encontrará que el segmento verde es la URL (Localizador Uniforme de Recursos) y el recurso solicitado está definido mediante el segmento azul URI.

Los recursos en la interfaz de programación se componen de un "namespace" y el método HTTP mediante el cual se ha de acceder. Dicho método recibe como parámetro la URI y el método especifico en la API.

GET: recupera información desde el origen especificado.

FMFServer.get ('/checkin/findCheckin/:username', checkinManager.findCheckIn);

POST: envía nueva información a la fuente especificada.

FMFServer.post ('/users/createUser',usersManager.createUser);

PUT: actualiza información existente de la fuente especificada.

FMFServer.put ('/users/updateUser', usersManager.updateUser);

DELETE: elimina información existente desde el origen especificado.

FMFServer.delete ('/users/deleteUser/:username', usersManager.deleteUser);

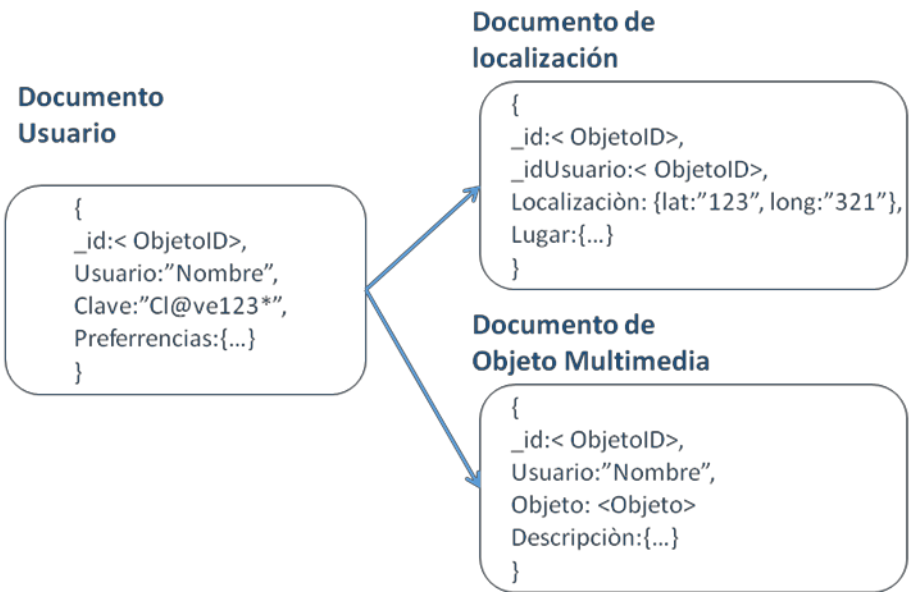


Figura 4.4: Modelado de datos, Estructuración de documentos.

Para el modelado de los datos se ha usado un modelo de objetos empleando JSON (JavaScript Object Notation), dado que es un formato de intercambio de datos ligero. Se basa en un subconjunto del lenguaje de programación JavaScript, estándar ECMA•262 3 ª Edición • Diciembre 1999.

JSON es un formato de texto completamente independiente del lenguaje pero utiliza convenciones que son familiares para los programadores de lenguajes C, C + +, C #, Java, JavaScript, Perl, Python, y muchos otros. Estas propiedades hacen que JSON sea un lenguaje ideal de intercambio de datos y se ajuste perfectamente a los requerimientos del sistema, haciendo que el servidor pueda manipular los datos de manera cómoda y precisa.

JSON se basa en dos estructuras:

- Una colección de pares nombre / valor. En varios lenguajes, esto se define como un objeto, registro, estructura, diccionario, tabla hash, lista con clave, o una matriz asociativa.
- Una lista ordenada de valores. En la mayoría de los lenguajes, esto se realiza como una matriz, vector, lista o secuencia.

Se han definido tres objetos JSON construidos a partir de los requerimientos funcionales del aplicativo y como resultado de un análisis de alto nivel, recopilando toda la información necesaria que ha de emplearse tanto del lado del cliente como del lado del servidor.

La construcción de los objetos se planteó de una manera híbrida entre las dos estructuras en que se basa el estándar JSON como se muestra en la figura 4.3. Existe una parte de la estructura para todos los objetos donde se crea una colección a partir de pares nombre/valor y otra parte donde se incluyen listas ordenadas para almacenar información del mismo tipo perteneciente al mismo objeto.

Cada objeto tiene definidos algunos campos obligatorios o mínimos para que la información fluya de manera exitosa.

5 EJEMPLO DE APLICACIÓN

De acuerdo a la definición del dominio del problema realizada en el apartado 4.1.1, se han tenido en cuenta las necesidades generales allí citadas y se ha propuesto el siguiente caso específico, es importante aclarar que es un ejemplo de uso es, un caso particular que sirve como herramienta para evidenciar el funcionamiento de la arquitectura ya que permite administrar diferentes tipos de objetos multimedia en un cliente especifico. En este caso es un cliente móvil, aunque podría ser cualquier otro como se muestra en el análisis del dominio.

Muchas veces alguien desea ir a comer a un lugar que no conoce, visitar nuevos sitios diferentes a los que siempre frecuenta o retroalimentar la calidad de algunos productos mediante puntuaciones y acciones interactivas que pueda compartir a través de un dispositivo móvil. Recopilando esta información, el propósito fue brindar a los usuarios la posibilidad de buscar restaurantes, teniendo acceso a su ubicación y la posibilidad de compartir su localización, desde su perfil de usuario. Los usuarios tendrán acceso a datos como la ubicación del restaurante, perfil o preferencias.

Para visualizar el aplicativo ver anexos apk (Código Client Android) o abra desde su celular la página https://testflightapp.com/

- ingrese con el usuario: testerfmf@gmail.com, contraseña: FmfTester123.
- Descargue la aplicación de testFlight siguiendo las instrucciones que le dan.
- Abra la aplicación e inicie sesión
-Descargue la aplicación la aplicación.

Las características principales son descritas en el siguiente apartado, donde se definen las características de uso.

5.1 REQUERIMIENTOS DEL SISTEMA

En este apartado se enuncian las características de uso, técnicas y funcionales, de la arquitectura limitada al dominio de problema específico o caso de ejemplo, el cual define unos requerimientos especiales que definen el funcionamiento del sistema.

5.1.1 Características técnicas

- Dispositivos móviles Android 17 conectados a internet y que posean una cámara.
- Uso de Frameworks que utilicen tecnologías web que permitan, a través de un solo desarrollo, abarcar múltiples plataformas en una aplicación empaquetada de manera nativa.
- Servidores capaces de resolver múltiples peticiones utilizando Node.js y bases de datos MongoDB.

5.1.2 Características de Uso

- El usuario empleará la aplicación cuando necesite conocer un restaurante nuevo y podrá ubicarlo mediante búsquedas por texto y geolocalización.
- El usuario podrá compartir su localización en los diferentes sitios que visite y de esta manera, ayudar a la aplicación y a otros usuarios en futuras búsquedas.
- Las preferencias de cada usuario influyen en el tipo de información presente en los diferentes tipos de búsqueda.

5.1.3 Características Funcionales

- Compartir localización en un sitio de comida.
- Tomar fotos del sitio y de los platos.
- Hacer búsqueda por texto.
- Hacer búsqueda por Locación.
- Crear usuario.
- Iniciar sesión Usuario.

5.2 PERSONALIZACIÓN DE LA ARQUITECTURA

El presente capitulo muestra cómo según la estructura general de la arquitectura, es particularizada de acuerdo a un caso específico y los requerimientos se ajustan al sistema construido, para controlar la información del ejemplo de aplicación y ver la funcionalidad de la arquitectura en un caso real.

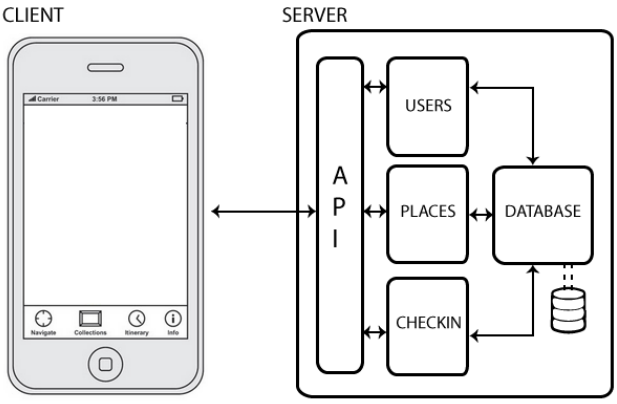


Figura 5.2: Arquitectura general personalizada.

5.2.1 Personalización del Modelado de datos

Para el modelado de los datos se hizo uso de los módulos inicialmente desarrolladlos en el sistema y personalizados según la figura 5.2, los siguientes apartados muestran como se ha definido los objetos que son enviados por el cliente y procesados por el sistema.

Usuario

```
var user = {
    _id:'n3nc8cn98nc8nc83',
    name:'Daniel Camilo Daza Diaz',
    username:'dazuku',
    password:'1234',
    gender:'male',
    typeUser:'Regular',
    profilePicture:'localhost/images/thumbnail.png',
    description:'soy re basilado',
    birthDay: new Date('Aug 22, 1991'),
    dateRegistration: new Date(),
    checkIns:[125627bhv6u98,827817928bsyyg9],
    list: [],
    friends : [
            {
                    friendId:"in5fn85n899n9d993d",
                    dateMakeFriendly: new Date()
            },
            {
                    friendId:"in5fn85n899ctjctkv5n9d993d",
                    dateMakeFriendly: new Date()
            }
    ],
    preferences: {
            foodKind:['chinese':true, 'Colombian':true, 'Thai':false],
            radius: 5
    },
    networks:[
            {name:'facebook', uniqueID:'123456', havePermitions:true},
            {name:'twitter', uniqueID:'123456', havePermitions:true}
    ],
    location:'Bogota, COL',
    email: 'dazuku.rulo@gmail.com'
};
```

Lugar

```
var place = {
    name:'El Corral',
    placename:'elcorral',
    logo:'localhost/images/thumbnail.png',
    branches:[2323eun4urbu4, 2y37y723h7hd72h4],
    dishes:['jnsjsnjn28273872','2988hd3h7h8h9'],
    premium:true,
    typesOfFood:['Thai', 'Chinese', 'Colombian'],
    ranking:5,
    characteristics:{
            homes:true,
            drive_in:true,
            internet:true,
            breakfast:true,
            vegetarian_food:true
    },
    comment:'hello world',
    popularity:4.5
};
```

Sucursal

```
{
        _id:123457,
        location:{point:{lng:'23', lat:'35'}},
        name:'Santafe',
        photos:[{date:1234567890, id:'1234567'}],
        dishes:['jnsjsnjn28273872','2988hd3h7h8h9']
}
```

Compartir localización

```
var checkIns = {
    date:'13 de septiembre de 2013',
    placeID:'nu7b3bubbbu3',
    branchID:'njnjn7y737279u29829s',
    userID:'neduhe737hd73gd73ghd7',
    placename:'elcorral',
    branchname:'colina',
    username:'dazuku',
    dishes:[
            {
                    name:'Hamburguesa',
                    photo:'localhost/images/thumbnail.png',
                    comment:'Sabrosa!',
                    ranking:5
            },
            {
                    name:'Hamburguesa de pollo',
                    photo:'localhost/images/thumbnail.png',
                    comment:'Sabrosa!',
                    ranking:5
            }
    ],
    comment:'hello world',
    photo:'/img/checkin.png'
};
```

5.3 DISEÑO

Para el diseño del ejemplo de aplicación se pensó en un diseño minimalista, y que cumpliera con todos los estándares de UI/UX por la plataforma Android (más información en http://developer.android.com/design/index.html), Para ello se propuso un flujo de pantallas, tal y como se muestra en la figura 5.3.

La pantalla número 1 es la pantalla de inicio, donde el usuario podrá crear su cuenta o ingresar con una cuenta previamente creada.

Las pantallas 2a y 2b se encargan de recolectar los datos necesarios para la creación de un nuevo usuario.

La pantalla 3 es el mapa donde se muestran los restaurantes que se encuentran cerca de la ubicación donde se encuentra el usuario, en esta pantalla el usuario puede buscar un lugar específico o hacer “Check-in” en el lugar que se encuentra.

La pantalla 4 da la opción al usuario de ver su perfil, aquí se muestra su información de usuario y los “Check-in” que ha realizado.

La pantalla 5 es mostrada al usuario cuando quiere hacer un “Chek-in”, aquí el usuario debe elegir el restaurante en el que desea hacer “Check-in”, si el lugar no existe el usuario tiene la posibilidad de agregar un nuevo sitio el cual lo llevara a la pantalla 7.

La pantalla 6 aparecerá en caso de que el usuario desee hacer una búsqueda de un sitio especifico, aquí solo debe ingresar el nombre del restaurante o sucursal que está buscando.

En la pantalla 7 el usuario podrá ingresar el nombre del restaurante y de la sucursal donde se encuentra en el momento de crear un nuevo sitio.

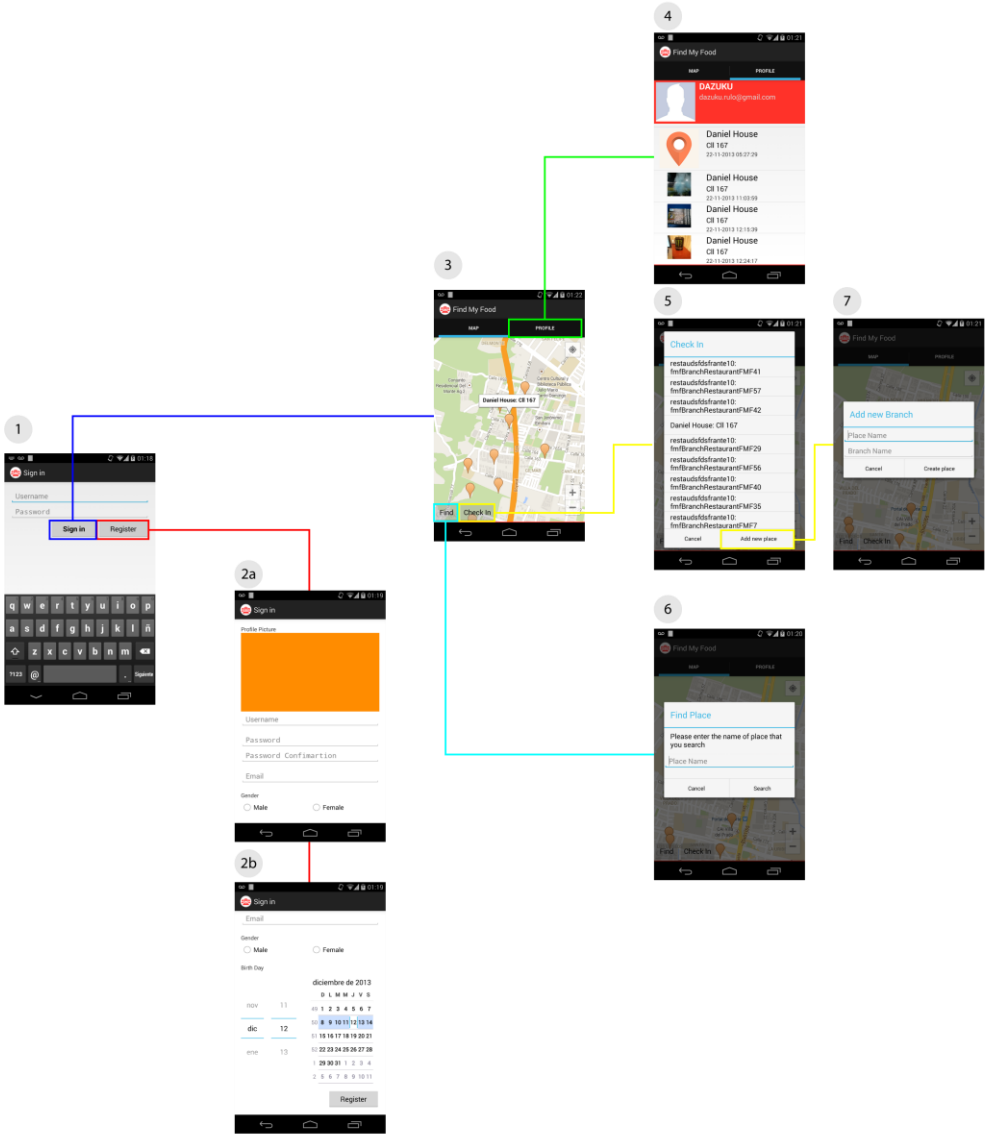


Figura 5.3: Diseño gráfico y flujo del ejemplo de aplicación.

6 RESULTADOS Y ANALISIS

Se ha obtenido una arquitectura de software modular y escalable capaz de insertar, actualizar, buscar y eliminar datos no estructurados mediante comunicación cliente-servidor, almacenando la información en una base de datos NoSQL y un prototipo de aplicativo desplegado en Android.

La arquitectura cuenta con un modelo de almacenamiento, como se especifica en el capítulo 4.2.1.1 y la Figura 4.2, que garantiza la escalabilidad del sistema. A medida que recibe peticiones, se crean hilos para responder a ellas, esto evita el problema de procesar las peticiones una a una y crear colas que desencadenan en mayor tiempo de espera en la respuesta. Los tiempos de respuesta por procesos están discriminados en cliente y servidor, estos datos los veremos con mayor detalle en el capítulo 6.6.

Se obtuvo también un modelo del aplicativo con un subsistema de comunicación soportado por REST para que el cliente pueda operar con el modelo del aplicativo.

La arquitectura cuenta con distintas funciones:

1 Administración del almacenamiento de datos
2 Transformación y presentación de los datos
3 Administración de la seguridad
4 Control de acceso a usuarios
5 Protocolo de acceso a la base de datos
6 Interfaz de programación de aplicaciones (API).

A continuación se mostraran los resultados de desempeño del sistema, los tiempos de respuesta de los diferentes clientes de prueba y se evidenciara que tan efectiva es la solución del problema planteado.

6.1 RESULTADOS DE DESEMPEÑO

Los resultados del desempeño tienen como objetivo mostrar los tiempos de respuesta y ejecución de los procesos que se realizan del lado del servidor y el cliente, y los tiempos que tarda la arquitectura en almacenar los datos para procesar la indexación, actualización, eliminación y búsqueda tanto de datos planos, como objetos multimedia.

Este tipo de resultados permite identificar el ritmo en el que puede crecer la información y el tamaño de almacenamiento de datos en el sistema, evidenciando la forma de escalamiento para determinar cuál tipo de cliente es más efectivo y de mejor respuesta.

La obtención de los resultados se dio a partir de pruebas en diferentes entornos; uno web y uno Móvil. El despliegue de la arquitectura se hizo sobre Nodejitsu, una plataforma que brinda el servicio para desplegar aplicaciones Node.js y permite implementar sin problemas este tipo de desarrollos en la nube, ofreciendo un conjunto de funcionalidades para ayudar en la gestión y despliegue de aplicaciones.

La conexión a internet para las pruebas fue de un ancho de banda de 5MB dedicados.

En el proceso de pruebas se hizo uso de cuatro tipos de clientes diferentes que posteriormente fueron agrupados en un cliente móvil y uno Web.

- Consola de comandos de Windows, Terminal Mac OS.

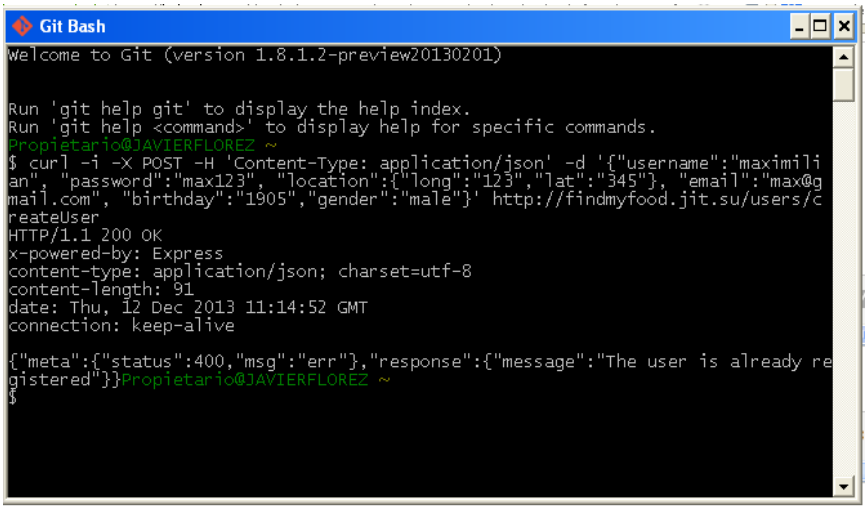


Figura 6.1: Envío de petición – Petición enviada desde consola de comandos.

- Cliente en un entorno Web “HTTP Dev Client”

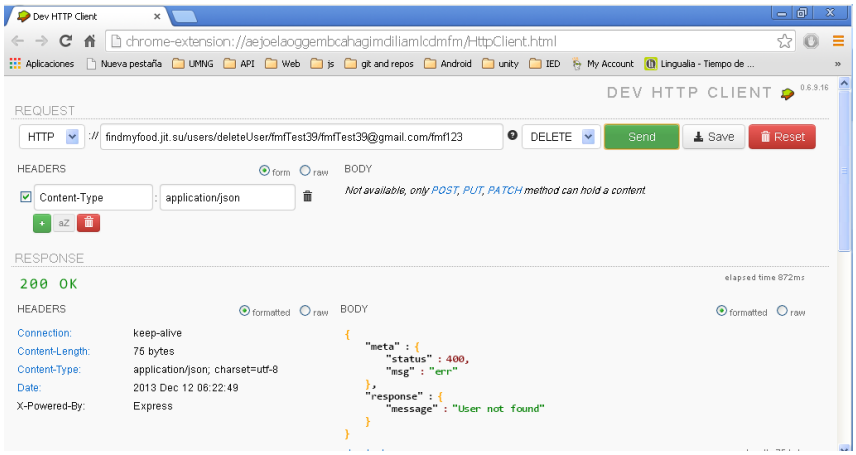


Figura 6.2: Envío de petición – Petición enviada desde un cliente web.

- Android 17 (Find My food): Aquí se puede observar parte de la aplicación creada para el caso de ejemplo, encontrando la interfaz de usuario creada y la respuesta de tipo JSON en una situación de búsqueda.

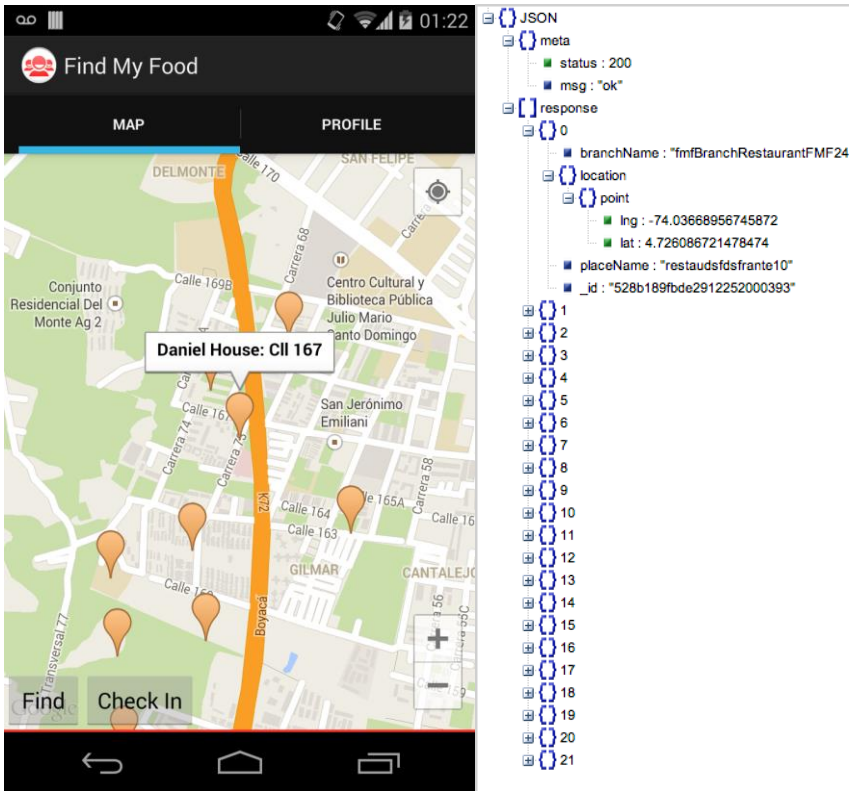


Figura 6.3: Envío de petición – Petición enviada desde un cliente Móvil.

A continuación veremos los resultados del proceso que consistió en enviar el mismo tipo de petición, recopilando los tiempos de respuesta tanto del servidor, como el cliente y los tiempos de procesamiento en el sistema.

Estos resultados fueron clasificados según el tipo de datos y procesos específicos.

6.1.1 Inserción de Datos

Datos como Texto

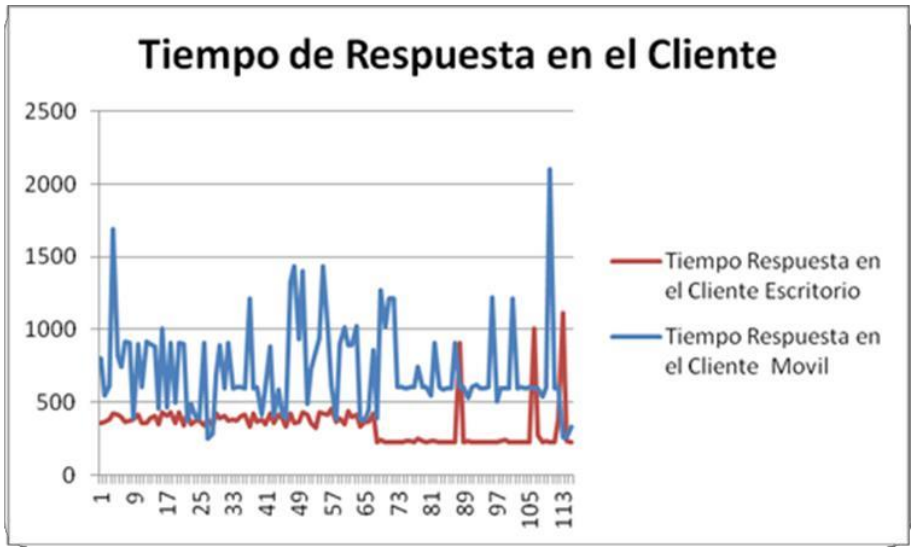


Figura 6.4: Inserción de Texto- Tiempo de respuesta en el Cliente, Tiempo/ Peticiones

Datos como imagen

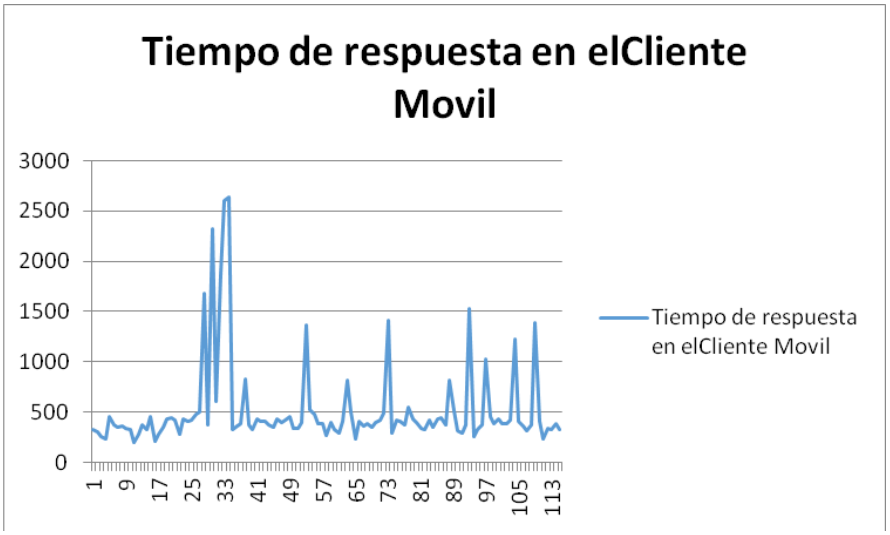


Figura 6.5: Inserción de Imagen - Tiempo de respuesta en el Cliente, Tiempo/ Peticiones.

En la tabla 4.1, ubicada en los anexos, se han registrado los resultados luego de evaluar el rendimiento en el proceso de inserción de texto y de imágenes. D icho rendimiento ha sido representado en las gráficas 6.4 y 6.5, la relación de dichas graficas está definida por el tiempo vs la cantidad de peticiones.

Se ha encontrado que los tiempos de respuesta en promedio son:

Inserción de datos como texto.
Cliente Web: 343 ms
Cliente Móvil: 722ms

Inserción de imágenes.
Cliente Móvil**:** 430ms

El rendimiento en el cliente de "escritorio" o cliente web es mucho más eficiente. Esto se debe a que la renderizaciòn de la información en el cliente móvil hace que el proceso tarde un poco más en completarse, teniendo en cuenta que los datos que se están mostrando vienen de un servidor en línea.

Se ha encontrado que el tiempo de inserción de imagen es mucho menor al tiempo de inserción de texto. Esto puede atribuirse a varias razones, pero la principal es que la inserción de datos pasa por un proceso de validación y verificación donde el modelo de la aplicación se asegura de que la información que se envía a la base de datos sea la correcta, esto con el fin de evadir sobre cargas para el sistema de almacenamiento con procesos que no cumplen las reglas establecidas.

6.1.2 Búsqueda de Datos

Búsqueda textual

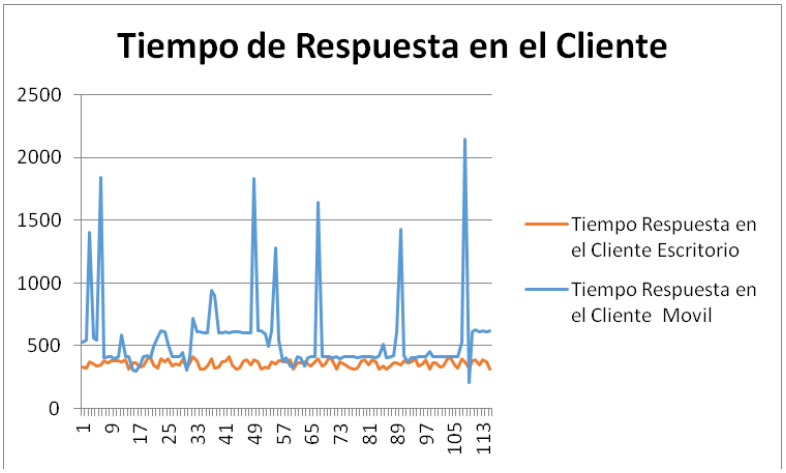


Figura 6.6: Búsqueda por texto - Tiempo de respuesta en el Cliente, Tiempo/ Peticiones.

Búsqueda por Geolocalización

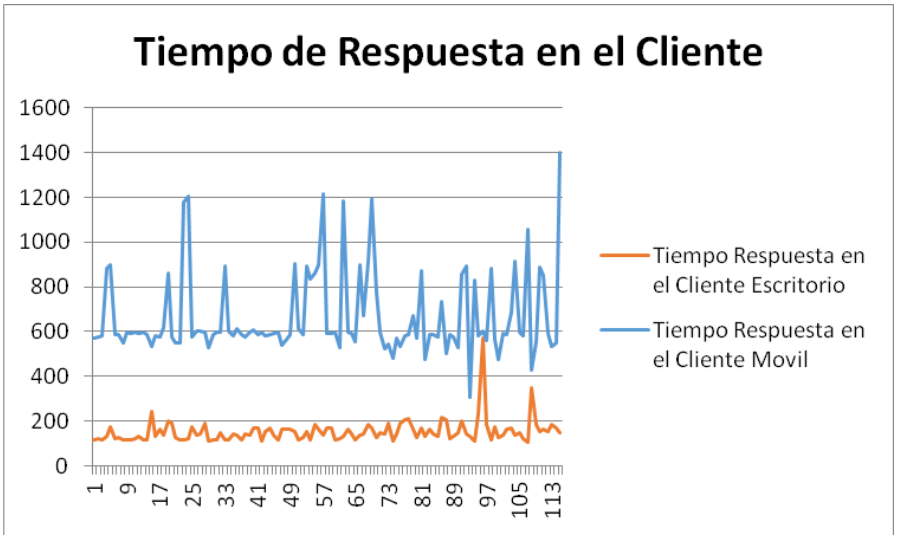


Figura 6.7: Búsqueda por Geolocalización - Tiempo de respuesta en el Cliente, Tiempo/ Peticiones.

De acuerdo a los resultados obtenidos en la tabla 4.2, se han generado las gráficas 6.6 y 6.7, los datos corresponden a la comparación de rendimiento en la búsqueda de datos desde un cliente móvil y uno web. La relación de dichas graficas está definida por el tiempo vs la cantidad de peticiones.

Se ha encontrado que los tiempos de respuesta en promedio son:

Búsqueda de datos por texto.
Cliente Web: 357ms
Cliente Móvil: 551ms

Búsqueda de datos por geolocalización.
Cliente Web: 152ms
Cliente Móvil: 665ms

Se encuentra que el tiempo de búsqueda por texto es casi la mitad del tiempo que emplea la búsqueda por geolocalización, esto se debe a que la consulta se hace de manera distinta ya que la búsqueda por geolocalización no es comparativa entre los registros de cada colección de documentos, si no que por el contrario, realiza un filtro según los parámetros enviados en la petición haciendo un poco más largo el proceso.

6.1.3 Modificación de Datos

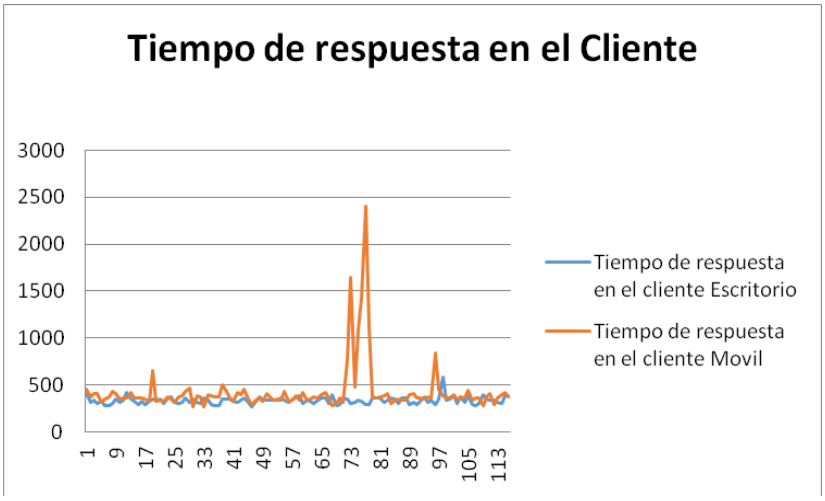


Figura 6.8: Modificación de Datos - Tiempo de respuesta en el Cliente, Tiempo/ Peticiones.

Teniendo en cuenta los resultados obtenidos en las pruebas de rendimiento que fueron registradas en la tabla 4.3 (Ubicada en los anexos), se ha generado la gráfica 6.8. Los datos corresponden a la comparación de rendimiento en la Modificación de datos desde un cliente móvil y uno web. La relación de dichas graficas está definida por el tiempo vs. la cantidad de peticiones.

Se ha encontrado que los tiempos de respuesta en promedio son:

Modificación de datos.
Cliente Web: 357ms
Cliente Móvil: 551ms

Observando de manera general, se puede ver que el tiempo de modificación es muy similar al de inserción de datos y evidentemente un poco menor. Esto se debe a que el documento en la base de datos ya ha sido creado previamente y las validaciones que se hacen al paquete de datos enviado desde el cliente, son mucho menores que las que se aplican a un paquete de datos enviado para ser insertado por primera vez.

6.1.4 Eliminación de Datos

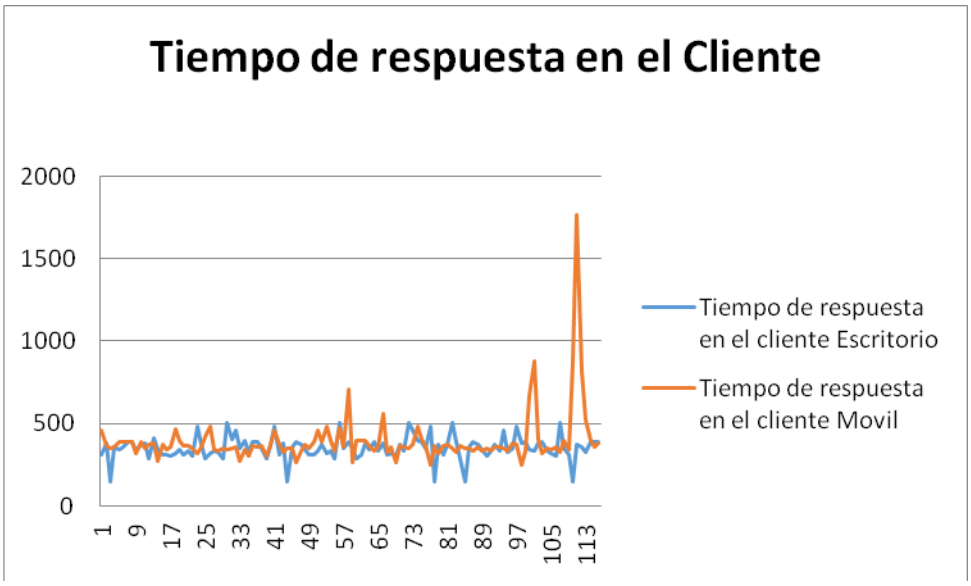


Figura 6.9: Eliminación de datos - Tiempo de respuesta en el Cliente, Tiempo/ Peticiones.

Luego de revisar los resultados de las pruebas en las cuales se evaluó el rendimiento del cliente en el proceso de eliminación y de ser registrados en la tabla 4.4 de los anexos, se ha generado la gráfica 6.9. Los datos corresponden a la comparación de rendimiento en la modificación de datos desde un cliente móvil y uno web. La relación de dichas graficas está definida por el tiempo vs la cantidad de peticiones.

Se ha encontrado que los tiempos de respuesta en promedio son:

Eliminación de datos.
Cliente Web: 395ms
Cliente Móvil: 353ms

Se encontró que el proceso de eliminación es el más similar en rendimiento entre cliente web y cliente móvil. Esto se debe a que la eliminación en los dos no requiere de renderización de datos, por ello el tiempo de respuesta que ha de suponerse para completar el proceso es bastante símil.

La eliminación es uno de los procesos, si no el más rápido, que se procesan del lado del servidor.

6.2 COMPARACIÒN DE RENDIMIENTO

Tiempo de Respuesta del Servidor

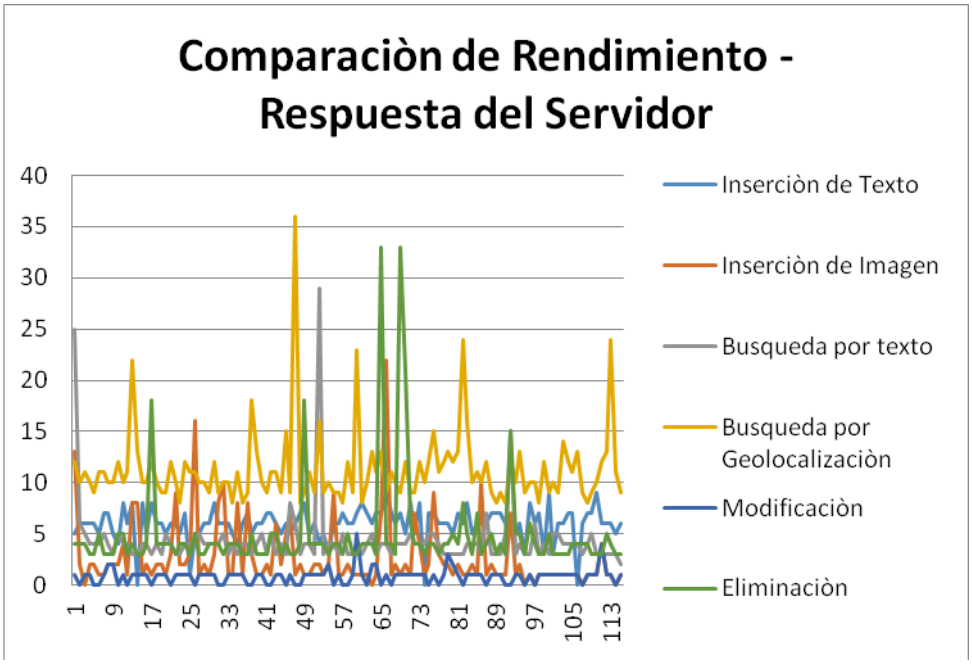


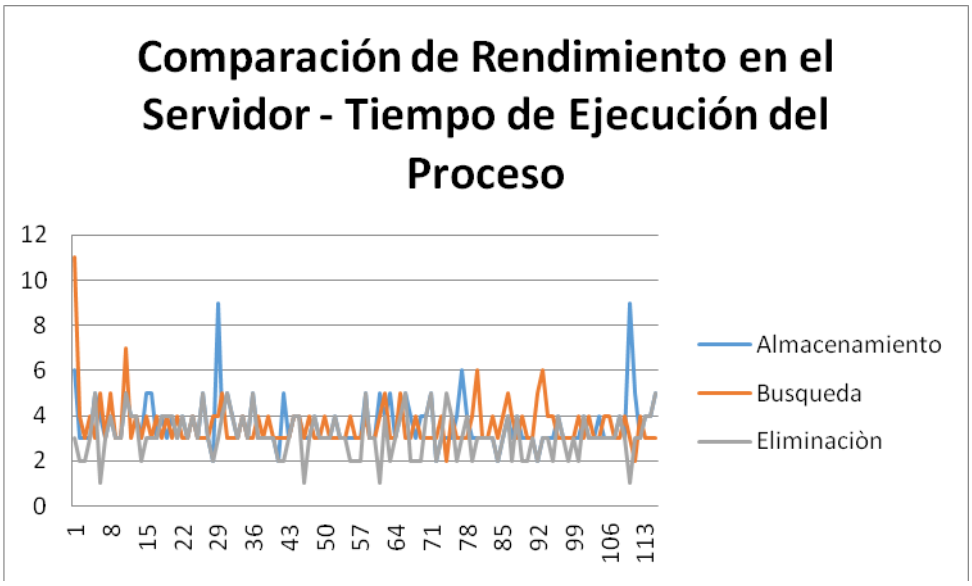
Figura 7.1: Comparación de Rendimiento en el Servidor - Tiempo de Respuesta.

La grafica 7.1 representa los datos consignados en la tabla 4.5 de los anexos, que fueron obtenidos luego realizar las pruebas de cada proceso en el servidor. La relación de la gráfica es tiempo vs. número de peticiones, el tiempo es netamente el que consume cada proceso en el servidor.

Cada proceso en el servidor gasta un mínimo de tiempo. Se puede ver que el proceso que más tarda es la búsqueda por geolocalización casi triplica al proceso de modificación, siendo este último el que menos tiempo consume.

Las respuestas enviadas por el servidor en promedio se mantienen sobre 8 ms. En general, cada proceso tiene un ritmo normal de desempeño y aunque en algunas peticiones se presentan algunos sobresaltos, esto puede atribuirse a peticiones que llegan simultáneamente y este tiempo de diferencia que se hace evidente. Esto corresponde a la creación de un nuevo hilo de conexión con el servidor, evitando los procesos en cola y bloqueantes.

Tiempo de Ejecución del Proceso en el Servidor



7.2 Comparación de Rendimiento en el Servidor - Tiempo de Ejecución del Proceso.

En la gráfica 7.2 obtenida a partir de los resultados registrados en la tabla 4.7, ubicada en los anexos, se evidencia que el tiempo que gasta el servidor en procesar la petición (este tiempo se genera desde el momento en que el modelo envía la información al sistema de almacenamiento, hasta que el sistema da respuesta la cual hace todo el flujo de información especificado en la gráfica 4.2) es un tiempo bastante corto.

Además, es importante resaltar que los sobre saltos que se presentan son bastante dispersos en el tiempo y que no representan una variación considerable en el tiempo de respuesta.

Tiempo de Ejecución del Proceso en el Cliente

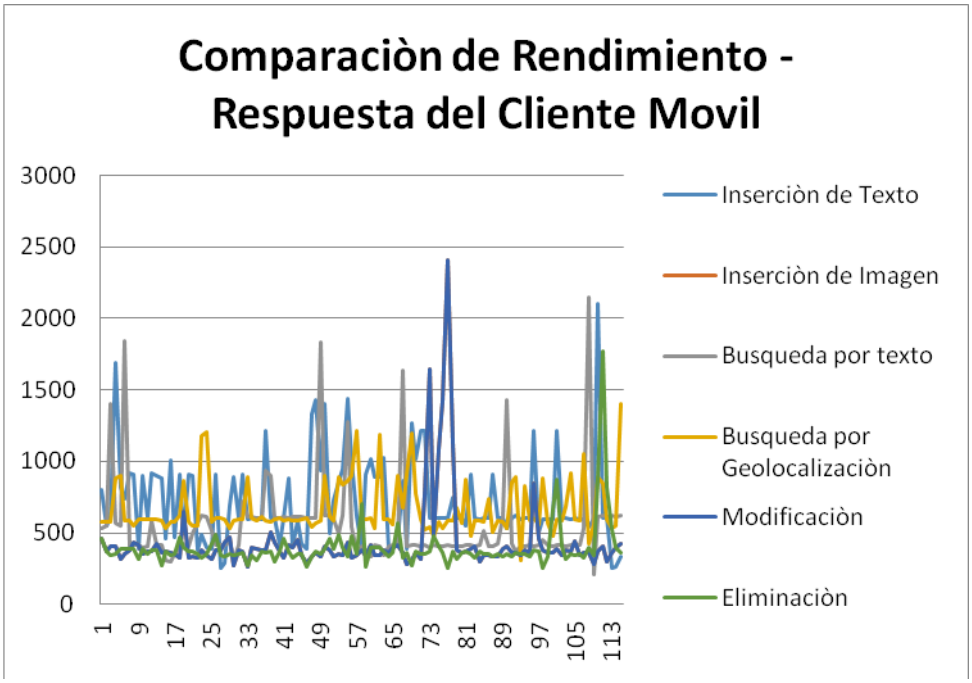


Figura 7.3: Comparación de Rendimiento en el Cliente Móvil - Tiempo de Respuesta.

A partir de la figura 7.3 se puede concluir que el proceso en el cliente móvil es el que más tiempo gasta desde que envía una petición y recibe una respuesta. Es de resaltar que dicha respuesta se da en un promedio medio segundo, lo cual es bastante significativo ya que se cumple con uno de los objetivos principales, obteniendo como resultado un servidor de rápida respuesta. Para consultar los datos con mayor detalle remítase a la tabla 4.8 consignada en los anexos.

Finalmente podemos concluir que el tiempo de respuesta del servidor puede variar dependiendo de la acción (insertar, actualizar, búsqueda o eliminar) que se desea realizar, esto se atribuye a factores externos como ancho de banda, memoria RAM y dispositivo de almacenamiento en el cliente. Por otra parte en los procesos básicos (insertar texto, insertar imagen, búsqueda de texto, búsqueda por geolocalización, modificación y eliminación) se puede ver que la petición más compleja es hacer una búsqueda por geolocalización y la menos compleja es la inserción de datos. Aunque existen operaciones más complejas que otras, la velocidad de respuesta de la operación más compleja es bastante rápida, lo que comprueba la efectividad de la arquitectura.

6.3 ANALISIS

Finalmente se obtuvo una arquitectura de software, que es capaz de almacenar y administrar grandes cantidades de datos de manera rápida y eficiente, todo esto gracias al diseño modular y uso de protocolos de comunicación.

Se muestra cómo el protocolo de comunicación cumple con los criterios básicos de calidad, encontrando que es un protocolo accesible, seguro y efectivo en el transporte de los mensajes en un tiempo reducido, siendo este uno de los principales objetivos planteados.

Al completar el desarrollo de la arquitectura encontramos que estas funciones obedecen a tres grandes componentes. Estos componentes contienen los módulos implementados (cliente, protocolo de comunicación, modelo de la aplicación, sistema de comunicación con la base de datos) y están definidos de la siguiente forma:

1. El componente de entrada/salida (I/O), el cual formatea y presenta los datos sobre el cliente Android que soporta el aplicativo móvil. La lógica de entrada muestra una pantalla de bienvenida en espera que el usuario ingrese los datos o realice alguna acción y luego responde al ingreso de datos (En este componente, la aplicación utiliza lógica de presentación para manejar la interfaz gráfica de usuario y el formateo de los datos).

2. El componente de procesamiento: Este se refiere específicamente a1 código de la arquitectura de lado del servidor que valida los datos y verifica los errores. La lógica del componente de procesamiento representa las reglas de negocio y la lógica de manejo de los datos para recuperarlos y guardarlos. Por ejemplo, la lógica de procesamiento en un registro de la locación (Checkin) que genera un ingreso de registro de locación y datos del usuario, una actualización de los eventos de dicho usuario e ingreso de datos alternos.

La lógica de procesamiento realiza varias funciones, incluyendo el manejo de entradas y salidas. Complementa las reglas de negocio, el manejo de los flujos de información dentro de la arquitectura y proyecta las transacciones reales en la base de datos mediante el módulo de comunicación.

Por consiguiente, el componente de procesamiento se puede dividir en tres subcomponentes funcionales desplegados en los módulos del modelo del aplicativo y el módulo de comunicación con la base de datos

a. La lógica de procesamiento de entrada/salida: valida los datos ingresados y verifica los errores.
b. La lógica de negocio: se aplica por conducto del código que representa las reglas del negocio.

c. La lógica de manejo de los datos: determina qué datos se requieren para cada transacción de negocio.

3. El componente de almacenamiento, que está compuesto por el módulo de comunicación con la base de datos, utiliza la lógica de manipulación de datos para encargarse del almacenamiento y recuperación de los datos en el servidor.

6.3.1 Clasificación del sistema cliente/servidor.

Para dar una clasificación al sistema cliente-servidor obtenido, teniendo en cuenta los estándares de clasificación, encontramos como resultado un sistema de cliente delgado, es decir, que se realiza un mínimo de procesamiento y del mismo modo, lo compone un servidor pesado donde se realiza la parte preponderante de la carga de procesamiento.

6.3.2 Arquitectura de Almacenamiento de Información

Se obtuvo un sistema de comunicación con la base de datos que se encarga de servir como administrador y proporcionar rutinas de comunicación especiales, que permiten que la base de datos acepte las solicitudes del usuario final. El sistema de comunicación dispone de funciones de transferencia de información, para tener acceso a la base de datos a través de internet por medio de un protocolo establecido desde el flujo de información del cliente hacia la capa del servidor.

Los usuarios finales pueden generar respuestas con la manipulación del cliente desde el aplicativo móvil construido para crear dicha interacción.

En cuanto a la administración, la arquitectura construida cuenta con estructuras necesarias para el almacenamiento de datos que liberan a1 usuario de la difícil tarea de definir y programar las características físicas de los datos.

La administración de la integridad de los datos está delegada en su mayor parte al sistema de comunicación con la base de datos y en una porción más reducida al modelo de la aplicación, quienes promueven y hacen cumplir las reglas necesarias para eliminar los problemas de integridad de los datos, reduciendo a1 mínimo la redundancia de los datos e incrementando a1 máximo la consistencia de éstos.

6.3.3 Modelo de la Aplicación

El modelo de la aplicación transforma los datos que se introducen de acuerdo con el formato y el protocolo dispuesto, necesario para guardarlos. Por consiguiente, el modelo libera al usuario de la tarea de distinguir entre el formato lógico y el formato físico de los datos. Es decir, formatea los datos físicamente recuperados para que se ajusten a las expectativas lógicas del sistema.

La interfaz de Programación de Aplicaciones (API) es pública con respecto a la aplicación en el cliente. El programador interactúa con el middleware mediante la API provista por la arquitectura. La API de middleware permite que el programador escriba un código genérico en lugar de un código específico para tener acceso a las posibilidades que brinda el sistema y así hacer uso de las funcionalidades provistas. En otras palabras, la API de la arquitectura permite que el proceso del cliente sea independiente del servidor, tal independencia significa que el servidor puede ser cambiado sin que tenga que volverse a escribir en su totalidad la aplicación en el cliente.

6.3.4 Protocolo de Comunicación

El diseño del protocolo de comunicación hace la labor de traductor para el servidor y para la base de datos. La capa del modelo del aplicativo acepta la solicitud genérica y la proyecta en el sistema de comunicación con la base de datos del servidor.

Como un servidor de base de datos podría tener algunas características no estándar, la capa traductora de la arquitectura puede optar por traducir la solicitud genérica al formato de datos.

7 CONCLUSIONES

La construcción de software para la gestión de información necesita mucho esfuerzo durante la fase de diseño y desarrollo. El caso de diseñar arquitecturas de software para moldear sistemas de información no estructurada, difiere de los modelos tradicionales de gestión de datos, el tipo de almacenamiento, la disposición lógica y física de los servidores, así como la comunicación entre cada parte del sistema. Este proceso ha llevado a materializar estas ideas de diseño sobre arquitecturas modulares ya que permiten que el software sea escalable, teniendo en cuenta el funcionamiento independiente de cada módulo, permitiendo el cambio de cualquiera de estos de manera rápida y sin restricciones mayores.

El ambiente cliente/servidor es fundamental para dividir los procesos de los dos actores, dándole mayor autonomía a ambos. La relación que sostienen el cliente y el servidor puede ser muchos a muchos donde un servidor proporciona servicios a muchos clientes, o uno a muchos donde un cliente puede solicitar servicios a muchos servidores, todo esto para optimizar el manejo de la información.

El manejo de grandes cantidades de datos multimedia es un proceso complejo que todavía es materia de investigación. Por esta razón, es óptimo hacer uso de nuevas tecnologías que soporten los últimos avances en este campo, permitiendo realizar desarrollos más robustos.

Con el buen uso de estas prácticas se ha obtenido una arquitectura de software Cliente/Servidor, con rápida respuesta, construida sobre estándares globales de desarrollo, programación, comunicación y almacenamiento para datos no estructurados.

Una arquitectura para la gestión de información ha de poseer varios módulos como el de almacenamiento, que aloja datos no estructurados y crea particiones según la cantidad de información que posee, evitando problemas de carga en el servidor.

El modelo de la aplicación, posee un módulo robusto para procesar las peticiones realizadas por clientes de diversos tipos, que es capaz de comunicarse bidireccionalmente entre la base de datos y dichos clientes.

Un protocolo de comunicación, REST o algún otro tan confiable como este, permite acceder de manera rápida a los recursos creados en el modelo del aplicativo brindando una API libre de ser usada en diferentes casos de estudio.

Una de las partes más determinantes en cualquier arquitectura son las bases de datos construidas a partir de datos lógicamente relacionados y guardados en un depósito de datos. El depósito de datos realiza el trabajo de guardar, introducir y manejar los datos para el

usuario final según la arquitectura que controla el flujo de la información. Es por ello que es de vital importancia contar con un sistema de administración de base de datos que se encargue de llevar a cabo estas operaciones.

El sistema de administración de base de datos potencializa la administración del sistema de información ya que permite eliminar la mayoría de los problemas que aparecen por inconsistencias de los datos, anomalías de los datos, dependencia de los datos y dependencia estructural.

Finalmente, la expectativa generada a futuro es lograr que se use el producto obtenido en diversos campos y disciplinas que requieran de manejo de datos sin una estructura definida. Además, se espera fortalecer la seguridad del sistema, incorporar minería de datos y búsqueda por patrones de imagen y audio, con el fin de garantizar su utilización en la mayoría de ámbitos que necesiten de la gestión de datos y desarrollo multimedia.

ANEXOS

Anexos Digitales

1. Manual técnico de instalación
2. Documentación del código
3. Léeme.
4. Código fuente del servidor.
5. Código fuente del cliente.

Tablas

4.1 Tabla de resultados inserción de datos- ./

4.2 Tabla de resultados búsqueda de datos - ./

4.3 Tabla de resultados modificación de datos - ./

4.4 Tabla de resultados eliminación de datos - ./

4.5 Tabla de resultados rendimiento del servidor - ./

4.6 Tabla de resultados tiempo del servidor procesando- ./

4.7 Tabla de resultados tiempo del cliente procesando- ./

Para hacer uso de la arquitectura desarrollada es importante seguir el manual técnico de instalación. **Ver Anexo 1.**

En caso de tener alguna duda sobre el funcionamiento de la interfaz de programación para la aplicación (API) puede consultar la documentación del código. **Ver Anexo 2.**

BIBLIOGRAFÍA

[1] Jeffrey Ullman, Prentice-Hall Inc., Simon & Schuster (1997). First course in database systems.

[2] Raul F Chong, Michael Dang, Dwaine R. Snow, Xiaomei Wang (3 Julio 2008). Introduction to DB2.

[3] Codd, Edgar F (Junio de 1970). Un modelo relacional de datos para grandes bancos de datos compartidos.

[4]Chapple, Mike. "Fundamentos de SQL". Bases de datos

[5] International Business Machines. . "Structured Query Language (SQL)".

[6] Tecnología de la información - Lenguajes de base de datos - SQL - Parte 1: Marco (SQL / marco)".

[7] Daniel Bartholomew. (Septiembre 2010) "SQL vs NoSQL". http://www.linuxjournal.com/article/10770. *Visitado ultima vez 20 Abril 2013*.

[8]."No SQL vs RDBMS databases" http://asktom.oracle.com/pls/asktom/f?p=100:11:0::::P11_QUESTION_ID:2664632900346253817. *Visitado ultima vez 20 Abril 2013.*

[9] Christopher C. Shilakes, Julie Tylman, (16 Novienmbre 1998) "Enterprise Information Portals".

[10] Webdevelopers notes. What is web server? http://www.webdevelopersnotes.com/basics/what_is_web_server.php *Visitado ultima vez 20 Abril 2013.*

[11] http://tomcat.apache.org/. *Visitado ultima vez 20 Abril 2013*.

[12] Klint Finley. (25 junio 2011). Wait, What's Node.js Good for Again? http://www.readwriteweb.com/hack/2011/01/wait-whats-nodejs-good-for-aga.php. *Visitado ultima vez 20 Abril 2013.*

[13] Jolie O'Dell (3 marzo 2011). Why everyone is talking about node http://mashable.com/2011/03/10/node-js/ *Visitado ultima vez 20 Abril 2013*.

[14] Alex Handy SDTimes. (24 junio 2012). "Node.js pushes JavaScript to the server-side". *Visitado ultima vez 24 abril 2013.*

[15] Implementations .http://wiki.commonjs.org/wiki/Implementations/node.js. *Visitado ultima vez 25 Abril 2013.*

[16].Express.js. http://expressjs.com/ *Visitado ultima vez 20 Octubre 2013.*

[17] Licesio J. Rodríguez-Aragón. "Internet y Teleinformática". .http://www.uclm.es/profesorado/licesio/Docencia/IB/IBTema4.pdf *Visitado ultima vez 25 Abril 2013*

[18] Fielding, Roy T. Gettys, James, Mogul, Jeffrey C. , Nielsen, Henrik Frystyk; Masinter, Larry; Leach, Paul J.; Berners-Lee (Junio 1999). "RFC 2616: Hypertext Transfer Protocol HTTP/1.1". http://tools.ietf.org/html/rfc2616

[19] Roy Fielding (20 Octubre 2008). API REST ". http://Roy.gbiv.com. *Visitado ultima vez 2 Septiembre 2013*

[20] "Estados de aplicación". www.tech.groups.yahoo.com. *Visitado ultima vez 2 Septiembre 2013*

[21] Trygve Reenskaug and James Coplien (20 Marzo 2009)."More deeply, the framework exists to separate the representation of information from user interaction." The DCI Architecture: A New Vision of Object-Oriented Programming.

[22] "The user input, the modeling of the external world, and the visual feedback to the user are explicitly separated and handled by three types of object." Applications Programming in Smalltalk-80(TM):How to use Model-View-Controller (MVC).

[23] Simple Example of MVC (Model View Controller) Design Pattern for Abstraction