PLATAFORMA CLOUD EXTENSIBLE DE MULTISERVICIOS INTEGRADOS PARA EL SOPORTE DE LA INFORMACIÓN DE VALOR AGREGADO PARA EL SERVICIO DE TRANSPORTE URBANO BASADO EN B.R.T.

CARLOS ANDRÉS PEREIRA GRIMALDO
JOSÉ GIOVANNI FLORES NOCUA.

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FÍSICO-MECÁNICAS
ESCUELA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA
BUCARAMANGA
2017

PLATAFORMA CLOUD EXTENSIBLE DE MULTISERVICIOS INTEGRADOS PARA EL SOPORTE DE LA INFORMACIÓN DE VALOR AGREGADO PARA EL SERVICIO DE TRANSPORTE URBANO BASADO EN B.R.T.

CARLOS ANDRES PEREIRA GRIMALDO
JOSE GIOVANNI FLOREZ NOCUA

Trabajo de grado para optar al título de Ingeniero de Sistemas

DIRECTOR
GABRIEL RODRIGO PEDRAZA FERREIRA,
PHD en Ciencias de la Computación

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FÍSICO-MECÁNICAS
ESCUELA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA
BUCARAMANGA
2017

DEDICATORIA

Dedico este trabajo a mi familia, que durante todo el transcurso de la universidad han estado incondicionalmente apoyándome física, moral y económicamente para terminar mis estudios, motivándome para seguir adelante y alcanzar mi meta profesional ya que sin su apoyo esto no hubiera sido posible, entre ellos:

A mi padre Isidro Pereira Morillo, a mi madre Luz Marina Grimaldo Cediel, a mi difunto abuelo Hernando Grimaldo, a mi abuela María del Carmen Cediel, a mis tíos Blanca Grimaldo y Jairo Medina, Álvaro Rojas Rondón, a Anggie Liseth Castellanos Camacho y los demás miembros de mi familia que no alcance a nombrar.

A mis tutores y maestros que se encargaron de mi formación y entender qué dirección tomar en mi carrera, entre ellos a mi maestro PhD. Gabriel Rodrigo Pedraza Ferreira, a mi maestro Fernando Antonio Rojas Morales. A mi maestro Elberto Carrillo Rincón. A mi maestro Manuel Guillermo Flórez Becerra, demás maestros que influyeron en mi formación académica.

A mis compañeros quienes han estado ahí para compartir conocimientos aprendiendo unos de los otros, entre ellos:

José Giovanni Flórez Nocua, Wilson Ramiro Prada Camacho, Ángel de Jesús Valbuena Navarro, Antonio Cortez, Rafael Sánchez, Jeremías Pabón, Fabio Montañez, Andrés Ferreira, Frank Díaz, todos mis compañeros de la plancha de centro estudios y el centro de estudios y los demás con los que tuve el gusto de conocer.

Y sobre todo a Dios.

Carlos Andrés Pereira Grimaldo.

Dedico mi labor a mis padres, siempre me estuvieron apoyando y brindando su sabiduría para formarme como persona de bien; a mis maestros, estuvieron dispuestos a brindar todo su conocimiento y experiencia para permitir mi formación profesional y como persona; a mis amigos y compañeros por apoyarme durante el desarrollo de mi carrera. Muchas Gracias por todo y que este proyecto sea una evidencia que todo ese apoyo no fue en vano, además de ser un punto de partida para la realización de mis sueños y lograr mis metas como persona.

José Giovanni Flórez Nocua.

CONTENIDO

TABLA DE TABLAS

LISTA DE FIGURAS

GLOSARIO

Apache-JMeter: Herramienta software que permite ejecutar pruebas de carga sobre el servidor.

API (Application Programming Interface): Es una interfaz software a software lo que permite que las aplicaciones puedan comunicarse entre sí.

BRT (Bus Rapid Transit): Bus de transporte rápido.

Eclipse (IDE): Entorno de desarrollo integrado.

Escalamiento horizontal: Consiste en agregar más nodos a un sistema para aumentar el rendimiento del mismo.

GitHub: Es plataforma de desarrollo colaborativo que utiliza Git como sistema de control de versiones.

HATEOAS: Consiste en utilizar hipermedia como motor del estado de la aplicación.

Hipermedia: Se basa en agregar información adicional a una respuesta de un servicio REST, que contiene una URI vinculada al recurso de interés, para que pueda ser consultado.

HTTP: Protocolo de transferencia de hipertexto; permite las transferencias de información en internet.

IoT (Internet of Things): Internet de las cosas; es una tendencia que busca que los objetos cotidianos poseen una conexión a la internet pretendiendo darle una identidad virtual a estos objetos.

JSON: Formato ligero de intercambio de datos.

MongoDB: Base de datos NO-SQL que almacena múltiples tipos de datos en documentos.

Real Time (Tiempo real): Hacer referencia a procesos computacionales que operan lo suficientemente pronto para considerarse como en tiempo real.

Stakeholders: Son todos los actores que se ven afectados por las decisiones de una empresa.

URI: Identificador de recurso uniforme.

RESUMEN

TÍTULO: PLATAFORMA CLOUD EXTENSIBLE DE MULTISERVICIOS INTEGRADOS PARA EL SOPORTE DE LA INFORMACIÓN DE VALOR AGREGADO PARA EL SERVICIO DE TRANSPORTE URBANO BASADO EN BRT*

AUTORES: JOSE GIOVANNI FLOREZ NOCUA
CARLOS ANDRES PEREIRA GRIMALDO**

PALABRAS CLAVE: Stakeholder, Platform Cloud, BRT, REST, JSON, MongoDB, HTTP

DESCRIPCIÓN:

En los últimos años la mayor parte de la sociedad se ha ido concentrando en las ciudades, según indica la ONU en el 2015 cerca de 54 % vivía en ciudades y se espera que para el 2030 esta cifra ascienda a 60 % [1], es por esto que se requiere que las ciudades tengan la capacidad de transformar sus sistemas y optimizar el uso de sus recursos a través de la tecnología. Se busca que la información esté disponible a todos los Stakeholders, de esta forma ellos puedan tomar decisiones que optimicen los recursos, es en este aspecto en el que se centra esta investigación, usar la tecnología para brindar información de valor agregado para uno de sus sistemas, el cual corresponde al servicio de transporte urbano basado en BRT.

En ese orden, para poder lograr que la información de valor agregado sea accesible a todos los Stakeholders que la necesitan, fue necesario crear una plataforma Cloud con un conjunto de servicios para acceder a esta información, además para desarrollar esta plataforma debe incorporar propiedades de software que garantiza la escalabilidad, rendimiento, tolerancia a fallos, entre otros aspectos que garanticen su calidad. En el proceso de elaboración se acordó elaborar los servicios según la arquitectura REST, persistiendo la información en una base de datos no relacional como MongoDB, además de otras herramientas que fueron necesarias para su construcción.

* Proyecto de grado

** Facultad de Ingeniería Físico mecánicas, Escuela de Ingeniería de Sistemas Director Gabriel Rodrigo Pedraza Ferreira,

ABSTRACT

TITLE: EXTENSIBLE CLOUD PLATFORM OF MULTISERVICES INTEGRATED FOR THE SUPPORT OF ADDED VALUE INFORMATION FOR THE URBAN TRANSPORT SERVICE BASED ON BRT*

AUTHORS: JOSE GIOVANNI FLOREZ NOCUA
CARLOS ANDRES PEREIRA GRIMALDO**

KEYWORDS: Stakeholder, Cloud Platform, BRT, REST, JSON, MongoDB, HTTP

DESCRIPTION:

In recent years, the most part of the society has been concentrated in cities. According to the UN indicates that in 2015 near of 54% lived in cities and it is expected that in 2030 this figure rises to 60% [1], Is for that cities are required to have the capacity to turn their systems and optimize the use of their resources through technology. It seeks to make information available to all Stakeholders, so they can make decisions that optimize resources, it is in this aspect that this research is centered, take the technology and provide value-added information for one of its systems, which refer to the urban transport service based on BRT.

In order, to be able the value-added information will be available to all Stakeholders who need it, it was necessary to create a cloud platform with a set of services to access this information, in addition to develop this system must incorporate software properties that guarantees scalability, performance, fault tolerance, among others aspects to assure quality of it. Therefore, the decision was made to elaborate our services per the REST architecture and persisting the information in non-relational database like MongoDB, besides other tools that was needed to build the platform.

* Project of grade

** Faculty of Engineering Physical Mechanical. School of Engineering of Systems Director Gabriel Rodrigo Pedraza Ferreira,

INTRODUCCIÓN

Una ciudad inteligente, es aquella que optimiza sus servicios y recursos básicos, para proporcionar mayor velocidad de respuesta a operaciones que al realizarlas con un esfuerzo humano tomaría mucho tiempo; para esto hay diferentes núcleos que apoyan a que una ciudad sea más inteligente. Entre esos núcleos se encuentra el sistema de transporte, el cual se encarga de movilizar a las personas de un lugar a otro. En Bucaramanga, Santander; existe un gran problema de movilidad, el cual, no solo se debe a la infraestructura de las carreteras, también se debe a la inserción de un sistema integrado de transporte que usa buses de tránsito rápido con un sistema de administración desactualizado, con información poco estructurada, ocasionando letargos en la solución de problemas pertinentes al sistema, con el fin de ayudar a solucionar el problema de administración, se ha decidido apoyar desde la parte informática a este sistema con una plataforma Cloud que brinda diferentes servicios para poder llevar a cabo un monitoreo, control y despliegue de información a diferentes Stakeholders, que no solo es capaz de realiza el trabajo básico del sistema, sino provee funcionalidades especiales y propiedades que permitirán evolucionar para adaptarse a futuras necesidades; sin embargo ¿Qué es Cloud o computación en la nube?

La computación en la nube, es un paradigma que consiste en prestar servicios desde una red o más comúnmente internet, construidos de tal manera que puedan ser utilizados por usuarios no expertos para cumplir con una tarea específica. La tendencia de utilizar computación en la nube, se debe, al hecho de usar servicios en la nube implica una reducción de costos de hardware y su mantenimiento, incrementa la productividad y la seguridad de una actividad, su construcción es económica y se apoya en software libre de primer nivel, implementación más rápida y con menos riesgos, el mantenimiento no implica tantos cambios, además

que el cliente está desacoplado de la plataforma cloud, entre otras ventajas. Sin embargo, una aplicación implementada en la nube, debe tener una conexión permanente a Internet y su velocidad de conexión debe ser de alta velocidad. Una forma de implementar servicios en la nube es por medio de REST, el cual, es una arquitectura de interfaces de comunicación que usa HTTP para la comunicación entre cliente y servidor.

Ahora bien, con ayuda de estas tecnologías fue posible realizar el proyecto que se presenta a continuación.

1. DEFINICIÓN DEL PROBLEMA

El crecimiento poblacional en todo el planeta se ha venido incrementando, cada vez la esperanza de vida es mayor y a su vez más personas siguen migrando a las ciudades buscando mejor futuro, motivo por el cual, resulta urgente solventar uno de los problemas que se desencadena directamente de dicho crecimiento, movilizarse entre dos puntos de una ciudad.

Colombia no es ajena a esta problemática, por ello se hizo necesario plantear un sistema de transporte que soporte este tipo de crecimiento sin colapsar. [4]Aunque existen varias alternativas como el metro, el tranvía u otros, resulta ser que estas soluciones suelen tardar considerablemente y ser bastante costosas. Es por estos motivos, los directivos de diferentes ciudades tomaron, en su momento, la decisión de optar por un sistema *BRT* que aparentemente combina la rapidez y eficiencia de un metro, con la flexibilidad de un bus, además de las reducciones en sus costos de implementación y operación, si solo si se dispone de la planeación necesaria para evitar que el sistema sucumba por la demanda.

Estos motivos llevaron a que se prefiriera el *BRT* sobre otros sistemas como el metro y el tranvía, aunque en un comienzo parecían soluciones que cumplían con los requerimientos de la demanda, con el paso del tiempo y la falta de planeación, la calidad del servicio se ha degradado considerablemente y presentan graves problemas en la mayoría de sistemas de transporte de las capitales Colombianas, generando un descontento general de los ciudadanos, que según la revista semana en Bucaramanga solo hay una conformidad del 15% con su servicio de transporte [4]. En mayor parte el descontento se debe a la falta de información que se tiene sobre el funcionamiento del sistema.

Para poder abordar la problemática de la falta de información de la prestación de servicios orientada a diferentes *Stakeholders*, se pretende implementar un sistema que proporcione información sobre el estado actual del servicio y que permita igualmente acceder a información histórica almacenada sobre la operación. La información proveída, está dirigida a diferentes *Stakeholders*, por una parte los usuarios del sistema *BRT* necesitan saber en qué momento puede abordarlo, que tanto tiempo va demorar su recorrido y conocer el estado de viaje e información de manera amigable, por otra parte los operadores de sistema deben tener la capacidad de monitorear el sistema para reaccionar rápidamente a los imprevistos que afecten la movilidad, los que toman las decisiones como alcaldías o gobernaciones, necesitan visualizar un panorama histórico y comprender el comportamiento del sistema y los terceros que requieran acceder a la información, por ejemplo, académicos o investigadores que tomarían datos históricos para realizar estudios sobre la movilidad de las ciudades.

Por este motivo, se evidencia la necesidad de proporcionar información a los *Stakeholders* para que puedan tomar acciones de acuerdo sus intereses particulares, de esta manera pueden mejorar la aceptación del sistema por parte de los usuarios.

2. OBJETIVOS

2.1 OBJETIVO GENERAL

Diseñar y desarrollar una plataforma Cloud con una variedad de servicios de acceso a la información que permitirá mejorar el funcionamiento del sistema de transporte urbano basado en *BRT*.

2.2 OBJETIVOS ESPECÍFICOS

- Diseñar la arquitectura de componentes Backend que se encargará de recibir e integrar los datos provenientes de diferentes fuentes referentes al sistema BRT.
- Proveer un conjunto de APIs que proporcione información sobre el estado del sistema a múltiples *stakeholders* y sectores para diversos usos.
- Implementar un prototipo software basado en la arquitectura planteada que incluya un subconjunto de componentes y APIS propuestos para la arquitectura Backend.
- Validar el prototipo funcional para verificar el funcionamiento de los componentes y APIS de acuerdo a los requisitos.

3. MARCO DE REFERENCIA

3.1 BRT

Un bus de tránsito rápido o por sus siglas en inglés *BRT* es un sistema de tránsito flexible, integrado y de alto desempeño. *BRT* combina la velocidad de los sistemas en rieles y la flexibilidad de los buses. El objeto de usar el *BRT* es emular las características de desempeño de un sistema masivo basado en rieles, pero mucho más económico que un tren ligero y aún más que un sistema de metro “Un sistema *BRT* normalmente va a costar de 4 a 20 veces menos que un sistema de tranvía o tren ligero y de 10 a 100 veces menos que un sistema de metro” [4]. Estos sistemas se componen de los siguientes elementos:

- Estaciones: Lugar apto para el abordaje y des abordaje de pasajeros.
- Carriles de autobuses: Sitios de desplazamiento de los buses.
- Plan de servicio: Mapa e indicaciones de rutas
- Sistemas inteligentes de transporte: Información que proporciona el sistema referente a información de paradas y prioridad en señalización de tránsito.
- Identidad e imagen del sistema: Son los colores, logotipos, nombres u otros elementos que diferencian el sistema de otro.

Alrededor del mundo se encuentra múltiples sistemas *BRT*, además no se dispone de un solo tipo de *BRT,* sino que por el contrario existen varias clasificaciones de este tipo de servicio, de acuerdo a un conjunto de requerimientos se determina a qué categoría de *BRT* pertenece. cabe resaltar que un *BRT* no solo se destaca por su velocidad o eficiencia en el servicio, sino también otros factores cualitativos resaltando que en un *BRT* completo hace énfasis en los clientes y es el modelo

ideal de los *BRT,* que se tratara en este caso donde la información es un actor importante. Esta situación se ilustra en la figura 4.

Figura 1. Servicios convencionales de transporte vs los sistemas BRT.

Fuente: ARIAS Cesar Guide spanish introduccion [en línea] disponible en: https://www.itdp.org/wp-content/uploads/2014/07/02.-BRT-Guide-Spanish-Introduccion.pdf referencia 4

Uno de los puntos críticos que pertenecen a un *BRT* son los sistemas inteligentes de transporte, en ellos se sustenta gran parte de la funcionalidad brindando información vital sobre el estado del sistema, que contribuye en la prestación de un servicio de calidad, en contraste a esta idea, los sistemas *BRT* que presentan un control de la información de forma artesanal sin presentar los mínimos requerimientos de calidad, impiden tener acceso a la información de forma **precisa y oportuna,** ocasionando un decremento en la calidad del servicio y por ende en los niveles de satisfacción. Llevar un control artesanal sobre la información en este ambiente es cómo grabar una película con una cámara de 2 Megapíxeles y esperar que los *Stakeholders* se conformen con verla en esta calidad.

Es por esto que es necesario presentar información de calidad y de forma oportuna, pero primero se necesita recopilar los datos necesarios para convertirlos en información útil, para ello disponemos en la actualidad de una tecnología conocida como *IoT,* que nos facilita la recopilación de información y de tecnologías en la nube logrando tener a disposición los datos recopilados.

3.2 ARQUITECTURA Y CALIDAD DE SOFTWARE

La arquitectura de software es la base fundamental de un software, porque en ella se describe de forma abstracta como es el sistema, como se debe comportar, como debe interactuar con otros sistemas, es la receta por la cual se construye una aplicación y depende de la capacidad del programador de crear un modelo del sistema para para su posterior construcción.

La arquitectura de un software se basa en requerimientos y restricciones, los requerimientos son extraídos del análisis de necesidades, estos son inspirados encuestas, entrevistas, experiencias de usuario, casos de uso, entro otros y las restricciones hacen parte de las limitaciones tecnológicas para implementar el sistema.

Este proyecto presenta una arquitectura la cual se basa en ofrecer servicios desde la nube, seguidamente se explicará una arquitectura para este fin.

3.2.1 Arquitectura orientada a servicios "SOA" SOA es una arquitectura para la construcción de aplicaciones de software construidas en base a servicios promoviendo la separación de sus componentes permitiendo el bajo desacoplamiento, de esta manera haciendo posible que el cliente pueda utilizarlos sin conocer su implementación. REST implementa SOA como arquitectura lo cual permite que este adquiera los beneficios que aporta SOA a una organización

como la optimización de procesos ayudando a que esta pueda adaptarse más rápido a los cambios

Para alcanzar atributos de calidad debe ser considerado en todo diseño, implementación y desarrollo. Los atributos de calidad no son totalmente dependientes de un solo factor, sino que por el contrario los resultados satisfactorios son una cuestión de conseguir la visión general (arquitectura), así como los detalles (aplicación) correctos,

además, se debe tener en cuenta que existen ciertos atributos de un sistema que pueden ser brindados por parte del mismo para garantizar la calidad. Todos estos pequeños detalles deben ser considerados para lograr un buen sistema que soporte la operación de transporte de los *BRT* y todos sus actores que participan en él. Algunos de los factores a tener en cuenta son:

3.2.2 Disponibilidad. Se preocupa por los fallos del sistema y sus consecuencias asociadas. Un fallo del sistema ocurre cuando este no es capaz de brindar un servicio según su especificación. A su vez se trata de como una falla es detectada y que tan frecuente ocurre y llegado el caso de ocurrir cuánto tarda en solucionarse. Este punto es crítico, puesto que es de vital importancia en un sistema que suministre información "*Real time"* implementado en un *BRT* se garantice la disponibilidad a la información todo el tiempo y a su vez se tenga un plan contra fallos, de tal forma que mitigue las consecuencias y no genere caos en el sistema y afecte no solo al flujo normal del mismo, sino que además afecte a otros usuarios ajenos al sistema.

3.2.3 Tolerancia a fallos. Se preocupa de que en la ocurrencia de una falla, un componente del sistema sea capaz de recuperarse y seguir prestando sus servicios sin afectar considerablemente el funcionamiento del sistema en general.

3.2.4 Rendimiento. Está relacionado con cómo el sistema es capaz de soportar grandes cargas de trabajo, cómo responde a las solicitudes de múltiples fuentes en simultáneo y cuánto tarda en responder. En un sistema *BRT* existen múltiples fuentes de datos. Por este motivo el sistema deberá ser capaz de responder lo más pronto posible a múltiples solicitudes de muchos *Stakeholders*, todo esto con el fin de garantizar que el factor rendimiento no se vea degradado.

3.2.5 Seguridad. Se trata de la capacidad de un sistema implementado en una plataforma Cloud puede resistir ataques que intenten quebrantarlo, tanto por un uso no autorizado o mal intencionado como de usuarios curiosos que prueba el sistema. Toda la información que se provee o ingresa a la plataforma debe ser controlada y la plataforma debe tener un modo de defenderse de los ataques al sistema de forma remota.

3.2.6 Usabilidad. Este factor trata de que tan fácil sea de “entender” el sistema, proporcionando la información para que otros sistemas o usuarios puedan acceder a ella y hacer uso de la misma, teniendo en cuenta edad, discapacidades u otros factores que puedan afectar al usuario objetivo.

3.2.7 Interoperabilidad Es la capacidad que tiene la plataforma de relacionarse con otras tecnologías o software permitiéndole funcionar independientemente de cómo se hayan implementado.

3.2.8 Escalabilidad. Trata de cómo el sistema que presta servicios a un *BRT* es capaz de tolerar el crecimiento de acuerdo a la demanda creciente de pasajeros por el crecimiento de las ciudades.

Todos los factores anteriores son necesarios si se desea tener una buena plataforma Cloud. Es necesario analizar cada uno de los factores mencionados puesto que en un sistema que prestará servicio a un *BRT* necesita información

pronta, verídica y sin fallas. Para ello el sistema estará dispuesto con una arquitectura hardware que soporte toda la carga y un software eficiente que no colapse por grandes cantidades de trabajo.

3.3 PLATAFORMAS EN LA NUBE E IOT

Computación en la nube puede ser definido como paradigma de datos y computación compartida sobre una red escalable de nodos (“La nube”) esos nodos incluyen clientes, centros de datos y servicios web (Mirzaei 2009). Betty define la computación en la nube como servicios basados en internet y recursos que son entregados a los clientes sobre demanda de un proveedor de servicio [8], es decir, las personas que hacen uso del sistema tienen acceso a diferentes servicios, los cuales son creado a partir de la necesidad del cliente por la organización, delegando los recursos como servidores, red, CPU, memoria y almacenamiento (Schulting, 2010).

La computación en la nube brinda la posibilidad a un sistema *BRT* de optimizar los recorridos mediante la adquisición de datos en tiempo real de las rutas por cada bus e integrarlos para crear información, por lo tanto permite planificar y mejorar la calidad de cada viaje y su sistema de información, en caso de que un sistema de transporte no posea un manejo adecuado de información a través de una infraestructura computacional el “*Cloud computing*” resulta como una ayuda significativa, ya que en contraste a adquirir una infraestructura computacional y los gastos que esto conlleva, se puede acceder a estos recursos por arriendo (como se hace con los servicios como la electricidad) reduciendo de manera significativa la inversión inicial [9], además que otorga una mayor facilidad para implementar los servicios que se le brindan a los *Stakeholders* del sistema. En últimas pagar por los servicios que proporciona la nube es más rentable que ir actualizando cada cierto tiempo toda la infraestructura tecnológica.

Los servicios de *Cloud* que se implementan en el sistema *BRT* son alimentados por administradores del sistema, *Stakeholders* mediante *Crowdsourcing* y buses inteligentes. La tecnología implementada en estos buses es denominada *IoT* forma parte de una tendencia que se ha presentado recientemente que pretende darles identidad virtual a objetos físicos, además de acceso a la *internetwork*.

Como ya se ha mencionado las plataformas *Cloud* reciben datos de múltiples fuentes entre los cuales se encuentran dispositivos *IoT*, para ello estas plataformas deben tener atributos software como la escalabilidad, rendimiento, confiabilidad, tolerancia a fallos entre otros conjuntos de características referentes a la arquitectura y calidad de software.

3.4 REST

Creada por Roy Fielding, *REST* es una arquitectura de interfaces de comunicación que utiliza el protocolo *HTTP*. Con esta arquitectura se pueden implementar servicios y aplicaciones independientes del tipo de plataforma o lenguaje que pueden ser accedidas por cualquier cliente que entienda el protocolo *HTTP.*

3.4.1 Características de REST.

- Protocolo cliente/servidor sin estado: cada petición *HTTP* contiene toda la información necesaria para ser ejecutada.
- HTTP: Las operaciones más importantes en *HTTP* son *POST*, *GET*, *PUT*, *DELETE*.
- Objetos en REST se manipulan a través de las *URI*: La *URI* es el identificador único de cada recurso.
- Sistema de capas: Arquitectura jerárquica entre componentes.

- Uso de hipermedia: Capacidad de una interfaz de desarrollo de aplicaciones de proporcionar al *Stakeholder* los enlaces adecuados para ejecutar acciones sobre los datos.

3.5 VENTAJAS DE REST

- Separación entre cliente y servidor: Permite que distintos componentes de un proyecto puedan evolucionar de forma independiente, además permite portabilidad entre plataformas.

- Visibilidad, Fiabilidad y Escalabilidad: La separación entre cliente y servidor permite que cualquier equipo de desarrollo pueda escalar el producto sin problemas. Esta separación convierte las aplicaciones en productos más flexibles a la hora de trabajar.[3]

3.5.1 Niveles de calidad de una API RESTFUL según Leonard Richardson:

3.5.1.1 Nivel 1. Uso correcto de las URI.

- Deben evitarse usar verbos en ellas.
- Deben ser únicas.
- Deben ser independientes de formato.
- Deben mantener una jerarquía lógica.
- Los filtrados de información de un recurso no se hacen en la *URI*.[5]

Ejemplo: http://localhost:8080/cloudBRT/api/monitoreo/buses/AMB123

3.5.1.2 Nivel 2: HTTP. Es el protocolo que permite las transferencias de información a través de la World Wide Web. Este dispone de métodos que nos

permite manipular los recursos disponibles, códigos de estado y tipos de contenido.

Ejemplo: Status: 200: OK.

Como se mencionó anteriormente las *URI* no deben contener verbos, aunque se quisiera manipular el recurso, para ello se encuentran los métodos *HTTP* que nos permite efectuar dicha tarea. Estos son:

- GET: Consultar y leer recursos.
- POST: Para crear recursos.
- PUT: Editar recursos.
- PATCH: Editar partes concretas de un recurso.
- DELETE: Eliminar un recurso.

A su vez toda acción sobre un recurso debe indicar una respuesta sobre cómo se dio dicha acción, si fue positiva o no, para ello se deben manejar códigos de estado que nos indiquen cómo se dio dicha tarea, para más información sobre cada uno de dichos códigos remitirse a [7]

Finalmente, HTTP permite recibir varios tipos de contenido que pueden ser indicados en un orden de preferencia en el *header*, a su vez al responder devolverá el mensaje según el primer formato indicado, de no poder mostrar el recurso en ninguno de los formatos se muestra el código de error HTTP 406, para terminar en la respuesta se devolverá el *header Content-Type*, para que el cliente sepa en qué formato se devuelve.

3.5.1.3 Hypermedia. Consiste en conectar mediante vínculos las aplicaciones clientes con las API, permitiendo a dichos clientes despreocuparse sobre cómo

acceder a dichos recursos, con este se añade información extra al recurso sobre cómo podemos acceder a otros recursos relacionados con el mismo.[4]

Ejemplo:


```
{
 "Agrego": true,
 "RecursoRuta": "http://localhost:8080/cloudBRT/api/monitoreo/rutas/P8",
 "RecursoParada": "http://localhost:8080/cloudBRT/api/monitoreo/paradas/ST1"
}
```


3.6 JSON

Es un formato de texto ligero de intercambio de datos, el cual está reemplazando a XML debido a que es más fácil crear un analizador sintáctico, además se facilita transcribirlo a un lenguaje orientado a objetos.

Ejemplo:


```
{
 "Clave":"ST19",
   "Nombre":"Est.Test",
   "Coordenada":{
   "Latitud":"1.0",
   "Longitud":"2.0"
   }
}
```

3.7 BASES DE DATOS NO RELACIONALES

Son bases de datos que se encargan de guardar grandes cantidades de información sin un esquema dado es decir no utilizan tablas, por consiguiente, no existe la operación join ni garantizan completamente ACID (atomicidad, consistencia aislamiento y durabilidad) y además son escalables horizontalmente, son llamadas también bases de datos orientadas a documentos ya que utilizan una estructura de objetos para almacenar información.[2]

Las bases de datos No Relacionales se crearon con la finalidad de abordar problemas donde es necesario una alta escalabilidad y rendimiento, para ello permiten una estructura de almacenamiento más versátil a costa de perder los atributos ACID mencionados anteriormente. [8]

4. ESTADO DEL ARTE

Normalmente, el manejo de la información de un servicio de transporte una ciudad con poco acceso y conocimiento tecnológico, aún es llevado por medio de planillas en las cuales una persona de forma manual anota la posición por la cual un bus ha pasado, para monitorear y controlar, así dejando un registro histórico privado que quizás llegue a ser desechado posteriormente.

Con el avance tecnológico, se crearon buses inteligentes que tiene la capacidad de generar información, entre la más relevante su coordenada, gracias a esto se han podido diseñar plataformas de uso específico para monitorear o suministrar información de una flota de buses en funcionamiento.

Recientemente las empresas que utilizan BRT (bus de tránsito rápido), tienen un sistema para llevar a cabo el monitoreo y el control desde una sala en la que hay personas encargadas de observar los inconvenientes desde un lugar cómodo y más seguro, como el caso de la empresa de carácter mixto Metrolínea, la cual lleva un registro físico en planillas y cuenta con un sistema de monitoreo, aunque algunos de estos sistemas ya se encuentran obsoletos.

Actualmente existe una filosofía llamada open data, que consiste en dejar ciertos datos para que puedan ser consultados por cualquier interesado ya sea académico, entidades del gobierno o de carácter científico; los datos de movilidad que deja como resultado meses de funcionamiento de un sistema BRT, pueden ser utilizados para realizar investigaciones acerca de la movilidad de una ciudad, estos pueden ser usados para mejorar la fluidez de las carreteras. Existe un repositorio de open data sobre la ciudad de Nueva York de libre consulta y entre las opciones hay una de movilidad.

Para concluir existen aplicaciones que son de uso específico tales como **Moovit** [11,12] que da información acerca de cómo moverse con el sistema de transporte de una ciudad o la iniciativa de una aplicación móvil cliente del grupo **GNET** de las Unidades Tecnológicas de Santander [13], la que permitirá que los usuarios conozcan la ubicación de un bus, aunque estas aplicaciones tienen un objetivo específico aún hace falta desarrollar un sistema completo que conglomere un conjunto de funcionalidades, además pueda seguir evolucionando para agregar más funcionalidades que una ciudad en su momento demande.

5. METODOLOGÍA

La metodología de trabajo que se llevó a cabo como un modelo por prototipos evolutivos, gracias a las características de este modelo, se podrán realizar modificaciones para mejoras futuras, además se puede ajustar para entregar un subconjunto de funcionalidades provenientes de dos ciclos de desarrollo a efectuar durante 16 semanas.

5.1 FASE 1: AMBIENTACIÓN TEÓRICA

- Investigación preliminar para el desarrollo de la plataforma.
- Lectura sobre tecnologías a usar para profundizar en el tema
- Capacitaciones sobre el uso de herramientas software.

5.2 FASE 2: IDENTIFICACIÓN DE NECESIDADES.

Por medio de la segunda fase se identificarán las necesidades generales y específicas que hay en el entorno de un sistema BRT. Se realizarán las siguientes actividades:

- Sondeo a pasajeros.
- Entrevista a Metrolinea.
- Análisis de datos.

5.3 FASE 3: ESPECIFICACIÓN DE ARQUITECTURA

En la tercera fase se propone la arquitectura software con sus características basándose en los resultados de la fase anterior. Durante el desarrollo de esta fase se tendrá en cuenta lo siguiente:

- Propuesta de la arquitectura
- Identificar las características de calidad para la plataforma
- Verificar la arquitectura propuesta.

5.4 FASE 4: IMPLEMENTACIÓN Y VALIDACIÓN.

Por medio de la cuarta fase se implementará el prototipo de acuerdo a las fases anteriores y se realizarán pruebas de simulación.

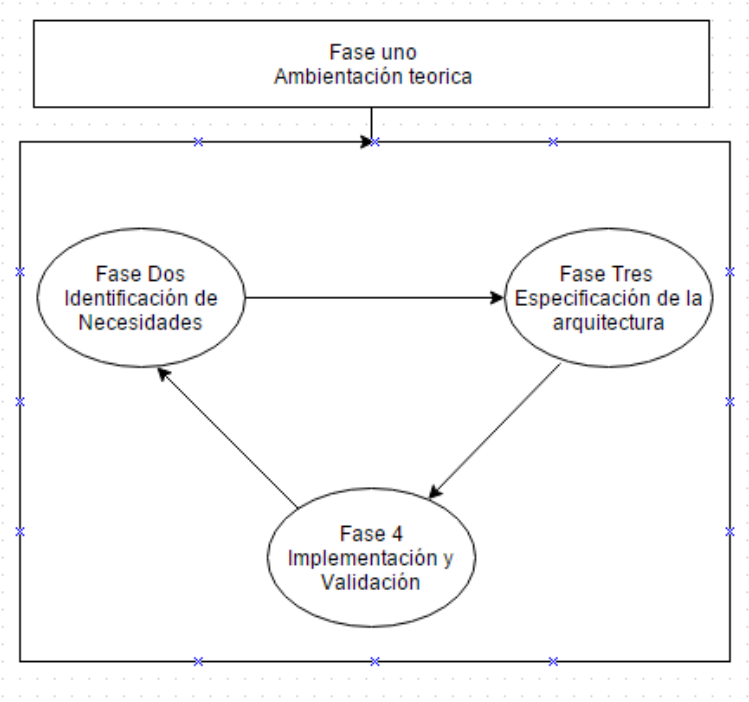
5.4.1 Ciclo 1: Codificación

- Codificación del prototipo
- Implementación de un subconjunto de servicios.
- Prueba con simulación de datos.

5.4.2 Ciclo 2: Optimización

- Resolución de problemas de la simulación.
- Actualización de servicios.
- Optimización de la plataforma.
- Pruebas finales y resultados.

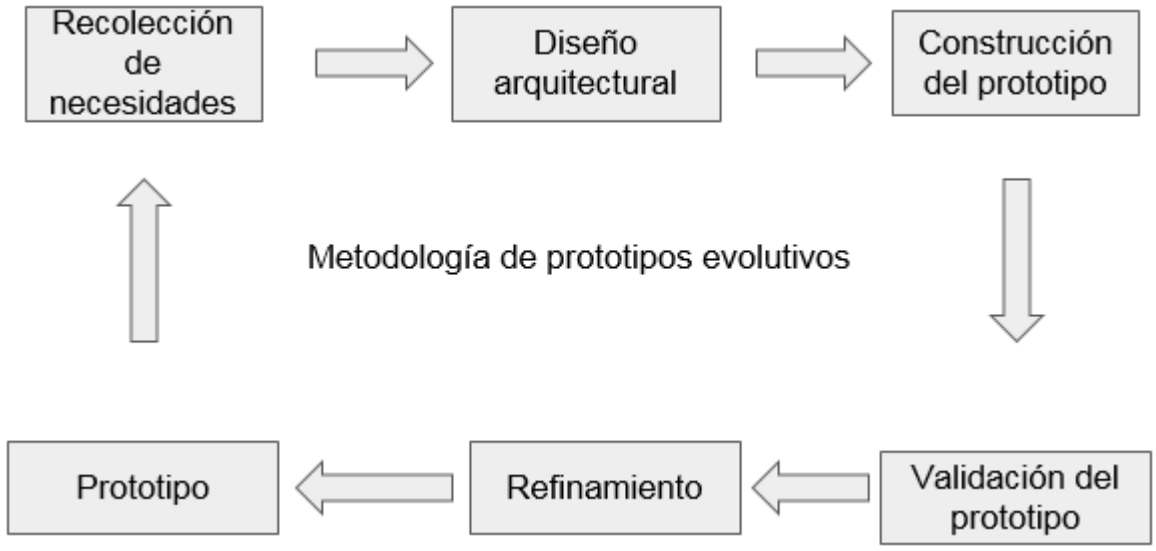
Figura 2. Fases del proyecto



6. DESARROLLO DEL PROYECTO

El desarrollo se llevó a cabo mediante el modelo por prototipos evolutivos, el cual permite crear un producto software por medio de ciclos, que es refinado hasta llegar a una versión más madura, que incluso puede seguir mejorando.

Figura 3. Metodología de prototipos evolutivos



6.1 NECESIDADES

Después de efectuar el proceso de búsqueda de necesidades, se identificó que en un sistema BRT se envían datos constantemente, todos los datos son recolectados, parcialmente tratados y visualizados únicamente para uso de la empresa, actualmente estos datos son convertidos en información básica para su funcionamiento, sin embargo, para mejorar el servicio es necesario agregar un conjunto de funcionalidades que le agreguen valor a la información, para que sea

clara y permitan la inclusión de stakeholders, como entidades del estado e investigadores, usuarios y administradores.

6.1.1 Funcionales:

- Se necesita que la plataforma provea información necesaria para realizar el monitoreo y control de un sistema BRT.
- Se necesita que la plataforma sea capaz de suministrar información en casi tiempo real y estáticas que permiten crear aplicaciones FRONT para los usuarios del servicio BRT.
- La plataforma debe almacenar información histórica que pueda ser consultada por investigadores y entidades del gobierno.
- Se necesita que la plataforma tenga la capacidad de manejar información relacionada con el georeferenciamiento justo como:
    - Recibir información proveniente de los buses
    - Calcular la distancia en línea recta entre 2 puntos geográficos.
    - Identificar y Registrar cuando un bus llega o pasa por una parada de su recorrido
    - Determinar el porcentaje de avance de un bus en un recorrido.

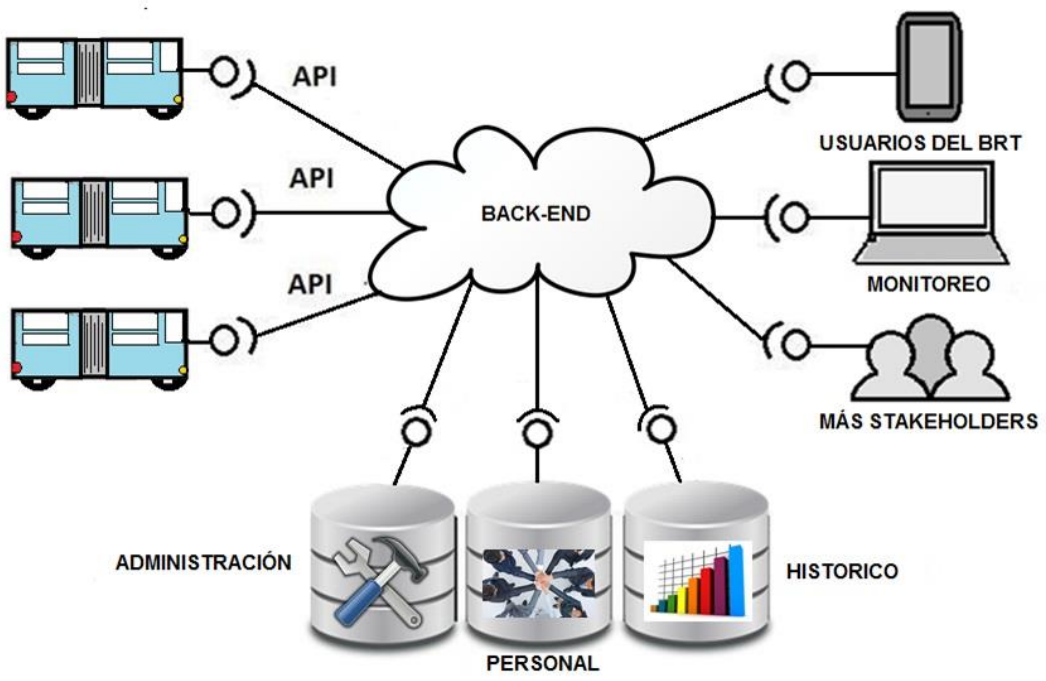
Para desarrollar las necesidades anteriormente descritas, se deben tener en cuenta unas propiedades no funcionales, entre las cuales se destacan:

6.1.2 No Funcionales: Esta claro que un producto software debe cumplir en gran medida con los requisitos no funcionales, sin embargo, hay que tener en cuenta uno de los objetivos del proyecto es crear un prototipo funcional, ya que este prototipo no es muy maduro no cumple de forma completa con estas propiedades, pero se pueden destacar 3 propiedades en las cuales se enfatizó el proyecto de forma inmediata, ellas son: **Escalabilidad**, **rendimiento**, **interoperabilidad**.

6.2 ARQUITECTURA DE LA SOLUCIÓN

Durante la fase de desarrollo se planteó una arquitectura que fue abordada desde dos perspectivas lógica y técnica, ambas arquitecturas describen el comportamiento y la ejecución del proyecto respectivamente, sin embargo, hay que destacar que él proyecto de grado se encuentran dentro de la Arquitectura General de un Macro proyecto, en el cual existen dos proyectos de grado adicionales que se apoyan como miembros de la siguiente estructura:

Figura 4. Arquitectura General del Proyecto Arquitectura del proyecto



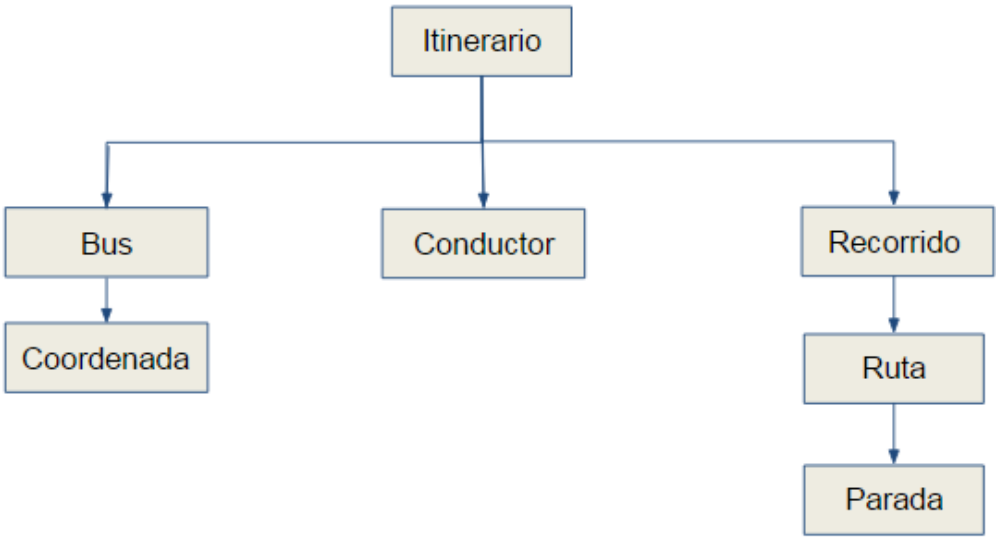
La imagen muestra que el prototipo backend se comunica por medio de interfaces, donde se alimenta de los buses para conocer sus datos procesarlos y lograr entregar información los diferentes usuarios posibles de la plataforma, también se puede evidenciar que la plataforma cuenta con interfaces a las bases de datos donde persiste los recursos del sistema como buses, paradas, rutas, recorridos

itinerarios y los conductores, además de ofrecer el almacenamiento de carácter histórico.

Ya entrando a la plataforma como tal se mostrará la arquitectura lógica que se encarga del comportamiento de la plataforma.

6.2.1 Arquitectura lógica: Se comenzó por diseñar los componentes base para que el sistema entienda todo el esquema de datos necesario para que la plataforma funcione de forma esperada; se estructuró un modelo de clases como base, que nos posibilita conocer una visión general del sistema, lo que nos ofrece un nivel de abstracción necesario para entender el funcionamiento de un BRT. El modelo de clases se planteó de la siguiente forma:

Figura 5. Diagrama de clases



Este diagrama de clase formo parte del módulo principal del sistema llamado procesamiento, el cual interactúa con los demás módulos que permiten que el prototipo trabaje como se espera.

Figura 6. Módulos de la arquitectura lógica

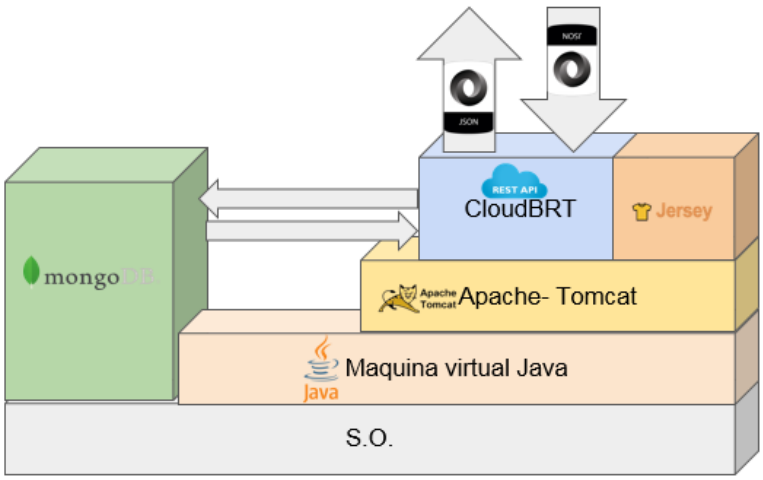
A continuación, se definirá la función de cada uno de los módulos del sistema y su interacción con los demás módulos.

- Módulo de recolección: En este módulo comienza todo el proceso. Se encarga la recopilar de los datos proveniente de los buses que cuentan con un pequeño tratamiento en el bus para ser recibidos con formato de datos que conoce la plataforma.
- Módulo de procesamiento: Es el modulo más grande y el que permite interactuar con los demás módulos del sistema, entre sus principales funciones este módulo pre procesa la información proveniente de los buses, aplica funciones de georeferenciamiento necesarias para relacionar al bus dentro del contexto del sistema, aquí se encuentra el modelo BRT en el cual se encuentran las abstracciones del diagrama de clases.
- Módulo de administración: Este módulo es el encargado de realizar las transacciones necesarias para administrar la información referente al sistema de BRT.
- Módulo de Persistencia: Contiene todas las clases que se encargan de interactuar con la base de datos para mantener la persistencia en la base de datos de los recursos y almacenar la información histórica
- Módulo de monitoreo: Contiene servicios que permiten conocer el estado actual de los recursos del BRT en tiempo casi real.

De forma general, el proyecto contiene un conjunto de servicios que reciben información de los buses, la procesa y genera información con valor agregado para los clientes FRONT-END, que no solo pueden consumir servicios de monitoreo y/o administración, también dar pie a aplicaciones FRONT futura para los usuarios del BRT. Para la persistencia de los datos, el Backend cuenta con respaldo de información estática, en bases de datos no relacionales, de cada uno de los componentes del sistema BRT como características de los buses, datos básicos de un conductor entre otros e información histórica del sistema que se espera sea de utilidad en futuros estudios.

Dentro del trabajo del prototipo fue necesario utilizar herramientas para su desarrollo, como apache MAVEN para la gestión y construcción del prototipo, Eclipse como entorno integrado de desarrollo y como lenguaje de programación Java junto con la intención crear un prototipo portable gracias a la Máquina Virtual de Java evitando compilar el prototipo para diferentes sistemas operativos y de fácil mantenimiento al trabajar componentes modulares con el estándar Java EE; El uso de estas herramientas dio como resultado un prototipo enmarcado dentro de una arquitectura física con más elementos.

Figura 7. Arquitectura física del prototipo



Como se puede observar en la Figura 13, se cuenta con un sistema operativo que contiene la Máquina virtual de Java (JVM), sobre ella Apache Tomcat para desplegar el prototipo CloudBRT que a su vez se apoya por librerías de Jersey con el fin de implementar REST para la comunicación mediante el protocolo HTTP, en donde se logra enviar en el cuerpo de los mensajes con el formato JSON, además se apoya la persistencia con MongoDB.

Ahora, todo el desarrollo del prototipo fue acompañado con el sistema de control de versiones **Github**, el cual nos permitió llevar cuantificar el volumen de trabajo para esta fase de desarrollo.

Figura 8. Volumen de trabajo en el desarrollo del prototipo

Commits durante el desarrollo del prototipo

Duración del desarrollo del proyecto: 5 meses y 25 días

En la gráfica se muestra una cresta de trabajo al comienzo que indica el esfuerzo en la implementación inicial de los componentes básicos para que el prototipo realice sus funcionalidades principales, posteriormente el trabajo decrece y se estabiliza, cuando se presentan valles es debido a que se realizaron validaciones y posteriormente se encuentran picos donde se realizaron mejoras pertinentes a estas validaciones.

6.3 IMPLEMENTACIÓN

Debido a que la metodología que se llevó a cabo fue por prototipos decidimos hacer dos ciclos así:

6.3.1 Ciclo 1: Codificación En el primer ciclo se realizó la codificación del proyecto, en el cual se liberó un conjunto de API básicas para los usuarios documentadas en un archivo con extensión .xls de Excel, las cuales fueron clasificadas de 3 formas según su función así:

6.3.1.1 Monitoreo. Es un subconjunto de *API´S* que se encargan de brindar información a usuarios del *front-end*, la cual advierte la ubicación de los buses, individualmente como los buses que se encuentran en un itinerario o recorrido dado, información acerca de las paradas, rutas, recorridos, itinerarios y conductores que llegaran a necesitar, además de conocer el porcentaje de avance de los buses entre paradas. Nótese que las peticiones de monitoreo tiene la característica de ser de consulta en su mayoría. Un ejemplo de un api de monitoreo es consultar un bus, utilizando el verbo GET en una petición HTTP para solicitar este recurso emitiría la siguiente respuesta:

Tabla 1. Api de Monitoreo Para Consultar un Bus

Petición	Entrada	Respuesta

http://localhost:8080/cloudBRT/api/monitoreo/buses/ABC123	-	{
		"Placa": "ABC123",
		"Capacidad": 123,
		"TipoBus": "Articulado",
		"Estado": true,
		"Coordenada": "
		{"latitud":0.0,"longitud":0.0}"
		}

6.3.1.2 Administración. Es un subconjunto de *API´S* que se encargan de administrar el contenido de la información del sistema, principalmente se encarga de realizar las transacciones a la base de datos y con ellas se puede crear, modificar y eliminar cada uno de los elementos que intervienen en el sistema como buses, rutas, conductores, paradas, rutas, recorridos, itinerarios. Para la administración un ejemplo de api es la creación de una ruta, devolviendo el siguiente resultado tras su consulta usando el verbo POST es:

Tabla 2. Api Administración Para Creación d Rutas

Petición	Entrada	Respuesta

http://localhost:8080/cloudBRT/api/admin/rutas	{
	“NombreRuta”: “P3”,
	“Categoria”: “Petroncal”,
	“Descripción”: “Recorre la zona este”
	}
		{
		"Creado": true,
		"RecursoRuta": "http://localhost:8080/cloudBRT/api/monitoreo/rutas/P8"
		}

6.3.1.3 Recolección. Este subconjunto de API se encarga de recoger la información dada por los buses acerca de su estado y ubicación, junto con registrar el histórico de cada uno de ellos. Un ejemplo de api para recolección es http://localhost:8080/cloudBRT/api/colector/buses usando el verbo POST requiere un cuerpo con la siguiente información:

Tabla 3. Api de Recolección Para Enviar Coordenadas

Petición	Entrada	Respuesta

http://localhost:8080/cloudBRT/api/colector/buses	{
	"Placa": "ABC123",
	"ProximaParada": 2,
	"Tde":"2016/10/16 13:13:00",
	"Coordenada":{
	"Latitud": "9.113633",
	"Longitud":"-72.114842"
	}
	}
		{
		"ProximaParada": 3,
		"Terminado": false
		}

6.3.1.4 Patrones de diseño. Durante la implementación del conjunto de API fue necesario utilizar algunos patrones de diseño que nos permitieron solucionar algunos problemas de orden lógico entre ellos se encuentran:

- Patrón de diseño Singleton: permitió solucionar el problema de mantener información en memoria RAM para modificar sus atributos y poder mostrarle al cliente los atributos actualizados que se requieran. Esto para evitar hacer uso de la base de datos lo que implicaría acceder al disco.
- Patrón de diseño Observer: permitió notificar a los itinerarios cuando un bus llega a una parada para modificar donde se encuentra el bus y establecer las paradas siguiente y anterior del mismo.

Finalmente, para realizar la validación de este ciclo se realizaron pruebas unitarias para corroborar que los servicios funcionaban correctamente según lo esperado, luego de eso se procedió a realizar pruebas de carga y rendimiento con ayuda de **apache Jmeter**, durante esta prueba se sometió el sistema a 100 peticiones cada 10 segundos, en este punto del sistema la prueba funcionó correctamente, sin embargo cuando se intentaba realizar más peticiones en esos 10 segundos el sistema se caía y dejaba de funcionar la plataforma, de esta forma se procedió al análisis del código para encontrar soluciones.

Una vez terminado el análisis, se concluyó que los problemas del servidor se debían a la plataforma no soportaba grandes cantidades de peticiones debido que se escribía repetidamente en disco y se llamaban repetidamente ciertas funciones, de esta manera se estaba generando carga a la plataforma.

6.3.2 Ciclo 2: Optimización En este momento comenzó el segundo ciclo, en el cual se realizó una optimización del sistema, donde estableció un parámetro de calidad de al menos 1000 peticiones cada 10 segundos como requerimientos mínimos y como requerimiento deseados 2000 peticiones cada 10 segundos. Dentro de este ciclo se actualizaron los subconjuntos de *API´S* de la siguiente forma:

6.3.2.1 Monitoreo. Se añadieron servicios de búsqueda adicionales, que permiten obtener información de la plataforma adicional, algunos se hicieron reciclando otras funciones para ofrecer más detalle.

6.3.2.2 Administración. En administración se creó un servicio para que los administradores del sistema puedan iniciar o reiniciar los itinerarios, de esta manera solo se movilicen los buses cuyo itinerario ha sido iniciado según el plan de movilidad de la empresa.

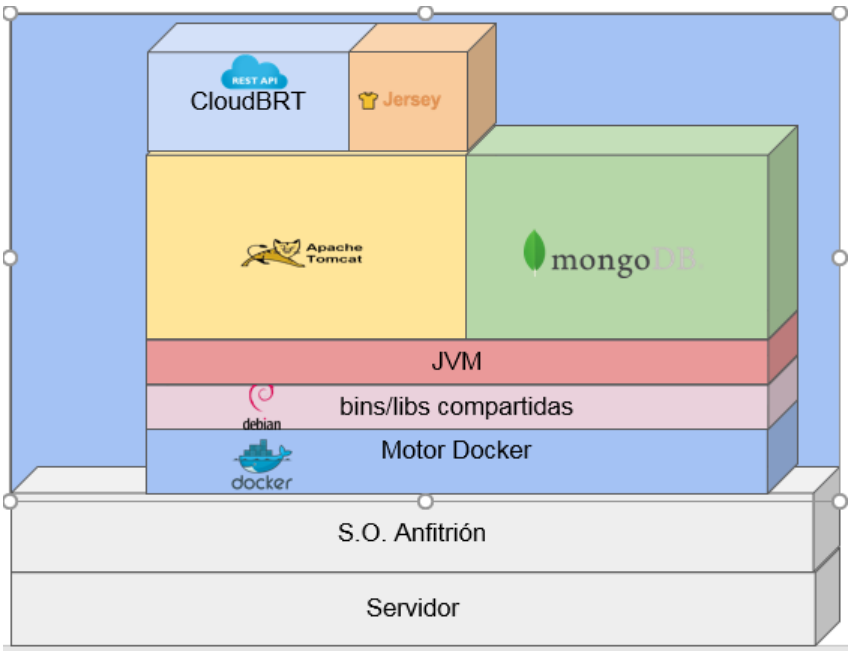
6.3.2.3 Recolección. Se removieron dos servicios pertenecientes a esta clase que no le corresponden. El primero de ellos se encarga de iniciar un itinerario que ahora pertenece a admin. El segundo se encarga de preparar el ambiente para la ejecución de la plataforma. Este servicio fue movido a la clase Arranque.

6.3.2.4 Arranque (nuevo). En este subconjunto se ubicarán las operaciones que deben inicializarse al inicio de la plataforma que le generan carga, para que de esta forma estén disponibles antes de comenzar el trabajo principal.

6.4 DESPLIEGUE

Era necesario desplegar el prototipo en un ambiente limpio y de forma rápida, debido a esto se optó por utilizar una tecnología llamada **Docker**, esta herramienta se puede instalar sobre un sistema operativo anfitrión en una maquina física o en máquina virtual, de tal forma que se pone en marcha su sistema de contendores, creados a partir de imágenes ligeras las cuales contienen las configuraciones necesarias para el despliegue automático la **arquitectura física del prototipo**, por medio de un documento de configuración de texto llamado Dockerfile.

Figura 9. Despliegue del prototipo sobre Docker



6.5 PROPIEDADES DE CALIDAD

En el desarrollo del proyecto se debe tener en cuenta aspectos de calidad, para esto se deben cubrir requerimientos no funcionales que son fundamentales para el correcto funcionamiento de la plataforma Cloud con gran demanda de información.

A continuación, se contemplarán las 3 propiedades no funcionales en las que se enfocó el proyecto:

6.5.1 Eficiencia El sistema tiene la capacidad de responder a las peticiones en un tiempo adecuado, permitiéndole tener la posibilidad de soportar una gran cantidad de peticiones sin degradar su calidad

6.5.2 Escalabilidad El sistema se ha construido para que pueda tolerar carga adicional de la que se espera en el servidor, es decir, si el cliente que usa la plataforma desea incrementar su flota de buses o quiere aumentar la cobertura de las aplicaciones adicionales puede hacerlo y el servidor seguirá ejecutándose sin problemas. Escalabilidad.

6.5.3 Interoperabilidad El sistema tiene un formato de datos sencillo que le permite interactuar con diferentes aplicaciones. Además, al ser una arquitectura orientada a servicios permite la implementación de sus funcionalidades de tal manera que se encuentra desacoplado del usuario que usara la plataforma.

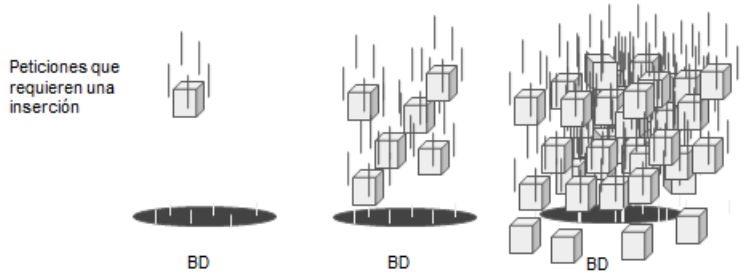
7. PRUEBAS FINALES

7.1 FASE PREVIA

Una vez se logró terminar el primer prototipo funcional se realizó una validación mediante una prueba de carga donde se enviaron 100 peticiones en 10 segundos al servicio de recolección en la que se perdieron el 100 % de peticiones.

Se encontró que este problema se debe a la necesidad de guardar en base de datos un registro histórico en el servicio de recolección de datos, debido a que el proceso de escritura en disco tiene un tiempo computacional elevado implica que la base de datos se sature al someter el servicio a grandes volúmenes de peticiones ocasionando una pérdida parcial o total de peticiones como se puede apreciar en la Figura 16.

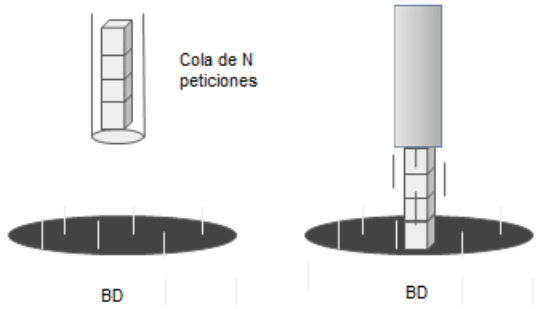
Figura 10. Ejemplo de pérdida de peticiones



Se desea que la plataforma soporte hasta 200 buses enviando peticiones como mínimo, sin embargo, el sistema debe tener la capacidad de escalar en caso de que un sistema supere los 200 buses, por este motivo fue necesario implementar una solución que evite la pérdida total de los paquetes, para esto se implementó una cola de peticiones que consiste en guardar hasta un tamaño N de peticiones

almacenadas en la RAM para posteriormente ser cargadas a la base de datos como se puede apreciar en la Figura 17.

Figura 11. Solución a la pérdida de peticiones



Un punto crítico para el prototipo es aquel servicio que requiere ser consultado repetidamente con acceso a la base de datos como el caso del servicio de recolección, se desea comprobar el rendimiento y la escalabilidad del prototipo actual. Se realizarán dos pruebas divididas en dos partes, sin embargo, para realizarlas es necesario conocer el ambiente en el que se realizaron las pruebas y el consumo de la aplicación.

7.2 CARACTERÍSTICAS DEL SISTEMA:

- 16 GB RAM, DRR3 1600 MHz
- Disco Duro SATA 7200 rpm
- QuadCore Intel Core i7 2600 segunda generación 3.5GHz

7.3 CONSUMO DE LA APLICACIÓN:

- Apache Tomcat con el WebService: entre 252 MB y 800 MB de RAM.
- JMeter: 1.7 GB de RAM
- Condiciones iniciales del WebService para la prueba:

7.4 PRIMERA PARTE: PRUEBA DE RENDIMIENTO “PERFORMANCE” PARA MEDIR EL PORCENTAJE DE ERROR POR NÚMERO DE INSERCIONES A LA BASE DE DATOS EN EL SERVICIO DE RECOLECCIÓN.

El objetivo de la primera prueba es descubrir el impacto de la escritura en disco sobre porcentaje de error, tomando como ejemplo 3000 peticiones en 10 segundos que afectan base de datos, donde se varía la cola que acumula peticiones para realizar una inserción a la base de datos. Se obtuvieron los siguientes resultados:

Tabla 4. Resultados de la prueba de Rendimiento

Cola de Peticiones	Inserciones Totales	% Error
10	300	98,43
20	150	96
30	100	89,6
40	75	82,57
50	60	71,07
100	30	71,07
200	15	13,7
300	10	10,6
500	6	7,53
1000	3	4,13

7.5 SEGUNDA PARTE: PRUEBAS DE CARGA

El objetivo de esta prueba comparar dos pruebas de carga en diferentes condiciones, para conocer bajo cuál de las dos situaciones el servidor responde mejor y que tanta carga de trabajo soporta en cada situación.

Se plantearon dos situaciones y para ambos casos enviaremos peticiones progresivas de 1000 en 1000 así:

- En la primera situación se variará la cola de peticiones a un cuarto del número total de peticiones.
- En la segunda situación, la cola de peticiones se mantendrá fija.

7.5.1 Primera situación: Prueba de carga con cola de peticiones a un cuarto de las peticiones totales. Se realizaron la prueba con 3 mediciones por cada 1000 peticiones por 10 segundos promediando el resultado, también se tuvo en cuenta la latencia para reflejar la carga sometida, de esta manera se obtuvieron los siguientes resultados:

Tabla 5. Resultados de la prueba de carga con cola de peticiones a un cuarto de las peticiones totales

Peticiones	Cola de Peticiones	% Error	Mínimo ms	Media ms	Máximo ms
1000	250	0	2	4	150
2000	500	0	1	4	149
3000	750	0	1	4	94
4000	1000	0	1	4	300
5000	1250	0	1	5	164
6000	1500	0,01	1	6	200
7000	1750	0,13	1	6	269
8000	2000	0,57	1	14	337
9000	2250	4,11	1	24	280
10000	2500	5,76	1	33	643

7.5.2 Segunda situación: prueba de carga con cola de peticiones fija. En este caso se estableció un tamaño fijo de inserciones cada 2500 peticiones. Se realizaron 3 mediciones por cada 1000 peticiones por 10 segundos promediando el resultado, también se tuvo en cuenta la latencia para reflejar la carga sometida, así se obtuvieron los siguientes resultados:

Tabla 6. Resultado de la prueba de carga con cola de peticiones fija en 2500

Peticiones	% Error	Mínimo ms	Media ms	Máximo ms
1000	0	2	6	281
2000	0,63	1	6	244
3000	0	1	4	161
4000	0,23	1	5	226
5000	2,6	1	11	217
6000	5,96	1	19	268
7000	9,3	1	28	269
8000	14,41	1	39	270
9000	19,34	1	48	312
10000	23,36	1	57	345

7.5.3 Comparación de las dos situaciones de la prueba de carga.

Figura 12. Comparación entre porcentaje.

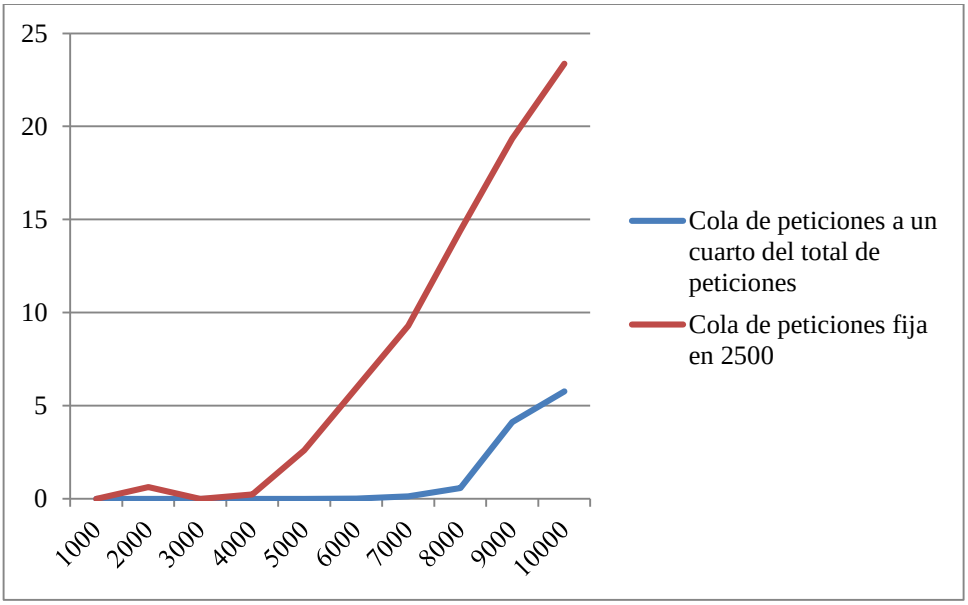
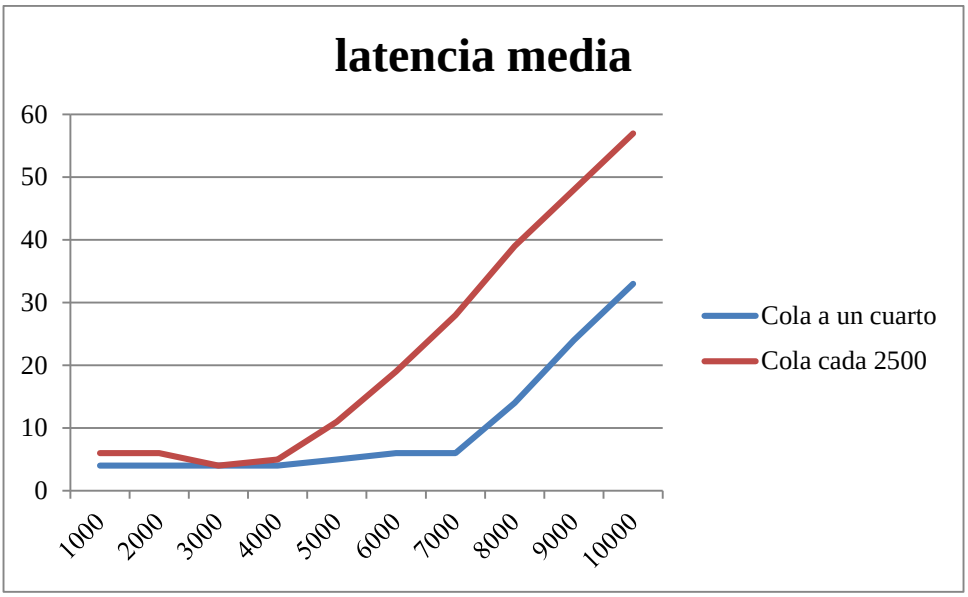


Figura 13. Comparación de latencias



7.6 ANÁLISIS DE LAS PRUEBAS.

- Como se puede observar en la tabla de resultado de la primera prueba, el porcentaje de error es inversamente proporcional a la cola de peticiones para realizar inserciones a la base de datos, esto se debe a que el tiempo computacional que se requiere para escribir en disco es mucho mayor que el que se requiere para escribir en RAM, por lo tanto, el procesador se ocupa de esa tarea volviéndose incapaz de atender las siguientes solicitudes.

- De la segunda prueba se concluyó que la mejor situación se da cuando la cola de peticiones se mantiene proporcional a la cantidad de peticiones enviadas a la plataforma, además la plataforma es capaz de tolerar aproximadamente 5000 peticiones en 10 segundos bajo las condiciones descritas sobrepasando fácilmente el volumen de 2000 peticiones en 10 segundos deseado.

8. CONCLUSIONES

- La plataforma cloud contribuye a mejorar la estructura de información para un sistema de BRT y que debe cumplir con un conjunto de requisitos como la escalabilidad, rendimiento y la interoperabilidad.
- Se Implementó un prototipo que contiene un subconjunto de las API´s que satisfacen la información necesaria y se diseñaron acorde a la arquitectura lógica.
- Se validó el prototipo mostrando su rendimiento y capacidad, de tal forma que se verificó el funcionamiento de sus API´s, enfatizando en las más críticas.
- Se concluye que un componente crítico en el sistema es aquel servicio requiera ser consultado repetidamente especialmente si requiere acceder a la base de datos, por ejemplo, el servicio de recolección de datos.
- En un sistema que maneja grandes cantidades de información, es importante controlar la forma en la que este hace registros a la base datos, para poder reducir el tiempo computacional que tarda el sistema procesando cada tarea, de esta manera, reducir el número de paquetes perdidos; se debe encontrar un equilibrio entre las peticiones entrantes y el número de escrituras a disco para cada sistema, sin embargo se puede evidenciar que al aumentar dicho valor ayuda a reducir el número de paquetes perdidos, pero como se mencionó anteriormente no debe aumentarse deliberadamente sino encontrar un equilibrio para cada sistema en específico.
- Es recomendable documentar cada uno de los servicios que se han implementado, debido a que los clientes necesitan una guía para entender inicialmente la estructura de los servicios cumpliendo así con un aspecto de calidad del software llamado usabilidad para que el usuario tenga un impacto amigable y de esta forma sea fácil de utilizar.

- Un conjunto de API RESTFul, debe cumplir con unos requisitos mínimos de calidad para garantizar un eficiente desempeño de la plataforma y poder reducir el consumo de recursos, al permitir ahorrar rentando servidores de mediano rendimiento.
- Una arquitectura orientada a servicios como RESTFul permite implementar un modelo cliente-servidor desacoplado, es decir, un servicio puede ser consumido independientemente de la naturaleza del cliente, permitiendo aumentar el alcance que tiene el proyecto a múltiples tipos de dispositivos.
- Ser consistente con la elaboración de las URI, permite que los clientes puedan intuir su estructura y reconocer como acceder a cada uno de los recursos de forma intuitiva reduciendo los tiempos de aprendizaje.
- Al trabajar con grandes cantidades de información, se requieren estructuras de datos eficientes, permitiendo que el servidor reduzca su tiempo de respuesta, aumentando así el número de peticiones que este puede atender, por este motivo se decidió emplear mapas de datos en lugar de arreglos, que a su vez permite obtener valores de forma rápida gracias a que trabaja sobre la estructura de datos interna y no sobre estructuras ordenadas en una capa superior.
- Gracias a la metodología de prototipos evolutivos, se escogieron herramientas y tecnologías además se escribió el código de tal manera que permita a la plataforma evolucionar dándole la capacidad de adaptarse a necesidades futuras no implementadas en este proyecto.

9. RECOMENDACIONES

- Dependiendo de las características del sistema, se debe efectuar un estudio sobre la proporción más eficiente entre peticiones e inserciones.

- Se encontró un mejor rendimiento en el servidor si se purga el histórico de los buses cada vez que el sistema inicia puesto que reduce el tiempo de carga. Para versiones futuras Implementar un script que lleve a cabo una limpieza del histórico de los buses la base de datos y los lleve a un archivo.

- Una vez el sistema esté en ejecución, debe iniciarse un servicio que prepara en entorno de trabajo para el funcionamiento de la plataforma. Para trabajos futuros, implementar una inicialización automática de estos recursos al iniciar el servidor.

- Debido a que la plataforma quedara abierta a nuevos ciclos para su evolución, se recomienda como siguiente paso agregar más API’s para el manejo del registro histórico.

REFERENCIAS BIBLIOGRÁFICAS

1. EL MUNDO Las megaciudades, llamadas a dominar el mundo [en línea] disponible en: http://www.elmundo.es/blogs/elmundo/entorno-habitable/2016/04/19/las-megaciudades-llamadas-a-dominar-el.html

2. MICROSOFT NoSQL vs SQL [en línea] disponible en: https://docs.microsoft.com/en-us/azure/documentdb/documentdb-nosql-vs-sql

3. Bbvaopen4u API REST: qué es y cuáles son sus ventajas en el desarrollo de proyectos [en línea] disponible en: https://bbvaopen4u.com/es/actualidad/api-rest-que-es-y-cuales-son-sus-ventajas-en-el-desarrollo-de-proyectos

4. ITDP Guía de ´planificación de sistema BRT Autobuses de transito rápido [en línea] disponible en: https://www.itdp.org/wp-content/uploads/2014/07/02.-BRT-Guide-Spanish-Introduccion.pdf

5. ASIERMARQUES Conceptos sobre APIS REST [en línea] disponible en: http://asiermarques.com/2013/conceptos-sobre-apis-rest/

6. REVISTA SEMANA Las ciudades medianas de Colombia viven un momento de auge [en línea] disponible en: http://www.semana.com/nacion/articulo/las-ciudades-medianas-de-colombia-viven-un-momento-de-auge/382710-3

7. VINA Y SAHNI Best practices for a pragmatic restful api [en línea] disponible en: http://www.vinaysahni.com/best-practices-for-a-pragmatic-restful-api

8. W3 Htresp [en línea] disponible en: https://www.w3.org/Protocols/HTTP/HTRESP.html

9. GEN BETA DE V El concepto Nosql o como almacenar tus datos en una base de datos no relacional [en línea] disponible en: https://www.genbetadev.com/bases-de-datos/el-concepto-nosql-o-como-almacenar-tus-datos-en-una-base-de-datos-no-relacional

10. JSRS: JAVA SPECIFICATION REQUESTS JSR 353: JavaTM API for JSON Processing [en línea] disponible en: https://jcp.org/en/jsr/detail?id=353

11. PÉREZ, María. MENDOZA, Luis. GRIMÁN Anna. Modelo para estimación de la calidad de un Web Service. Modelo para estimación de la calidad de un Web Service. *Universidad Simón Bolívar, Departamento de Procesos y Sistemas, Caracas, Venezuela.* 2009, nro 12, p..2-7.

12. MOOOVIT APP Localización [en línea] disponible en: https://www.moovitapp.com/?from=Ubicaci%C3%B3n%20elegida&to=Carrera%2027%20522-52104&fll=7.063075_-73.096418&tll=7.112603_-73.114702&metroId=4324&lang=es

13. VANGUARDIA LIBERAL Necesita conocer rutas de Metrolinea en Bucaramanga [en línea] disponible en: http://www.vanguardia.com/area-metropolitana/bucaramanga/385794-necesita-conocer-rutas-de-metrolinea-en-bucaramanga-esta-aplic

14. VANGUARDIA LIBERAL Diseñan aplicación para saber en dónde va el bus en Bucaramanga [en línea] disponible en: http://www.vanguardia.com/area-metropolitana/bucaramanga/381290-disenan-aplicacion-para-saber-en-donde-va-el-bus-en-bucaramang

15. IBM Arquitectura orientada a servicios (SOA): simplemente, un buen diseño [en línea] disponible en: https://www-01.ibm.com/software/co/solutions/soa/

BIBLIOGRAFÍA

ASIERMARQUES Conceptos sobre APIS REST [en línea] disponible en: http://asiermarques.com/2013/conceptos-sobre-apis-rest/

BBVAOPEN4U API REST: qué es y cuáles son sus ventajas en el desarrollo de proyectos [en línea] disponible en: https://bbvaopen4u.com/es/actualidad/api-rest-que-es-y-cuales-son-sus-ventajas-en-el-desarrollo-de-proyectos

EL MUNDO Las megaciudades, llamadas a dominar el mundo [en línea] disponible en: http://www.elmundo.es/blogs/elmundo/entorno-habitable/2016/04/19/las-megaciudades-llamadas-a-dominar-el.html

GEN BETA DE V El concepto Nosql o como almacenar tus datos en una base de datos no relacional [en línea] disponible en: https://www.genbetadev.com/bases-de-datos/el-concepto-nosql-o-como-almacenar-tus-datos-en-una-base-de-datos-no-relacional

IBM Arquitectura orientada a servicios (SOA): simplemente, un buen diseño [en línea] disponible en: https://www-01.ibm.com/software/co/solutions/soa/

ITDP Guía de ´planificación de sistema BRT Autobuses de transito rápido [en línea] disponible en: https://www.itdp.org/wp-content/uploads/2014/07/02.-BRT-Guide-Spanish-Introduccion.pdf

JSRS: JAVA SPECIFICATION REQUESTS JSR 353: JavaTM API for JSON Processing [en línea] disponible en: https://jcp.org/en/jsr/detail?id=353

MICROSOFT NoSQL vs SQL [en línea] disponible en: https://docs.microsoft.com/en-us/azure/documentdb/documentdb-nosql-vs-sql

MOOOVIT APP Localización [en línea] disponible en: https://www.moovitapp.com/?from=Ubicaci%C3%B3n%20elegida&to=Carrera%2027%20522-52104&fll=7.063075_-73.096418&tll=7.112603_-73.114702&metroId=4324&lang=es

PÉREZ, María. MENDOZA, Luis. GRIMÁN Anna. Modelo para estimación de la calidad de un Web Service. Modelo para estimación de la calidad de un Web Service. *Universidad Simón Bolívar, Departamento de Procesos y Sistemas, Caracas, Venezuela.* 2009, nro 12, p..2-7.

REVISTA SEMANA Las ciudades medianas de Colombia viven un momento de auge [en línea] disponible en: http://www.semana.com/nacion/articulo/las-ciudades-medianas-de-colombia-viven-un-momento-de-auge/382710-3

VANGUARDIA LIBERAL Diseñan aplicación para saber en dónde va el bus en Bucaramanga [en línea] disponible en: http://www.vanguardia.com/area-metropolitana/bucaramanga/381290-disenan-aplicacion-para-saber-en-donde-va-el-bus-en-bucaramang

VANGUARDIA LIBERAL Necesita conocer rutas de Metrolinea en Bucaramanga [en línea] disponible en: http://www.vanguardia.com/area-metropolitana/bucaramanga/385794-necesita-conocer-rutas-de-metrolinea-en-bucaramanga-esta-aplic

VINA Y SAHNI Best practices for a pragmatic restful api [en línea] disponible en: http://www.vinaysahni.com/best-practices-for-a-pragmatic-restful-api

W3 Htresp [en línea] disponible en: https://www.w3.org/Protocols/HTTP/HTRESP.html