SPRINT 1 - EXPERIMENTO CREACIÓN Y CONSULTA DE FACTURAS NIDOO

1. **El grupo describe correctamente los estilos y tácticas de arquitectura implementados para satisfacer sus ASRs.**

El principal estilo arquitectónico seleccionado para la aplicación NIDOO teniendo en cuenta el ASR del atributo de latencia es un estilo Multi-tier con 3 tier y como complemento se utiliza el estilo MVC.

Los tier definidos son:

1. **Tier Balanceo de Carga:** Este tier es al que se conectan los usuarios desde su aplicación en este tier se aloja un proxy y un balanceador de carga, para atender las peticiones de los usuarios y dirigirlas al Tier Web.
2. **Web Tier:** Donde se alojan los componentes para la presentación de información al usuario y manejo de eventos. Este tier internamente usa un estilo MVC, para atender las interacciones de usuario.
3. **BackEnd Tier:** Donde se alojan los componentes de acceso a datos y la base de datos.

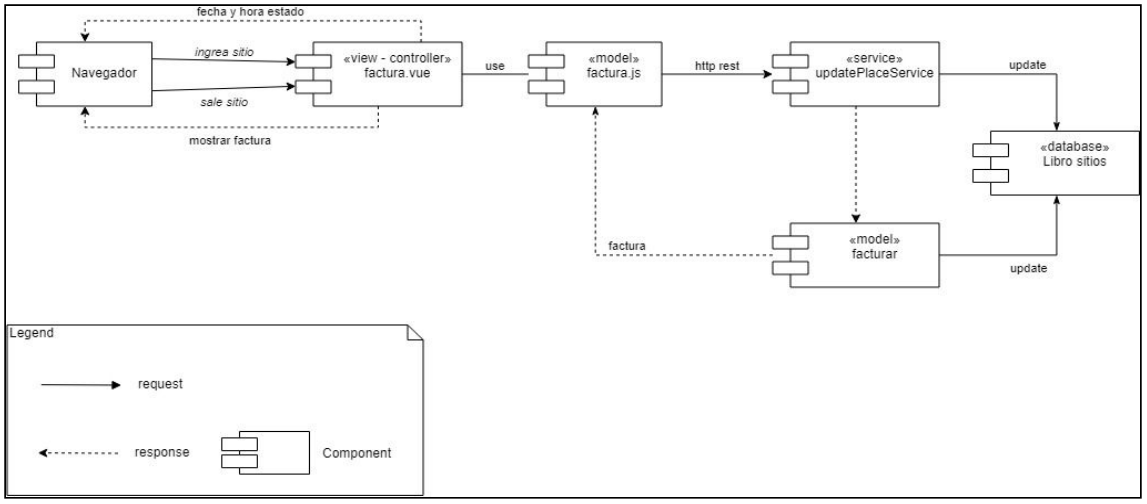
Los clientes se conectan al tier de balanceo de carga quien atiende y organiza las solicitudes, se encarga de pasarlas al tier web, que muestra información para el usuario, realiza cálculos, procesa los eventos e interacciones del usuario y obtiene información desde el Tier de Backend, quien se encarga de manejar el acceso a las múltiples bases de datos.

Tácticas Seleccionadas

Para garantizar el cumplimiento de los ASR de latencia seleccionados se opta por utilizar tácticas relacionadas con el manejo de los recursos, las tácticas utilizadas son:

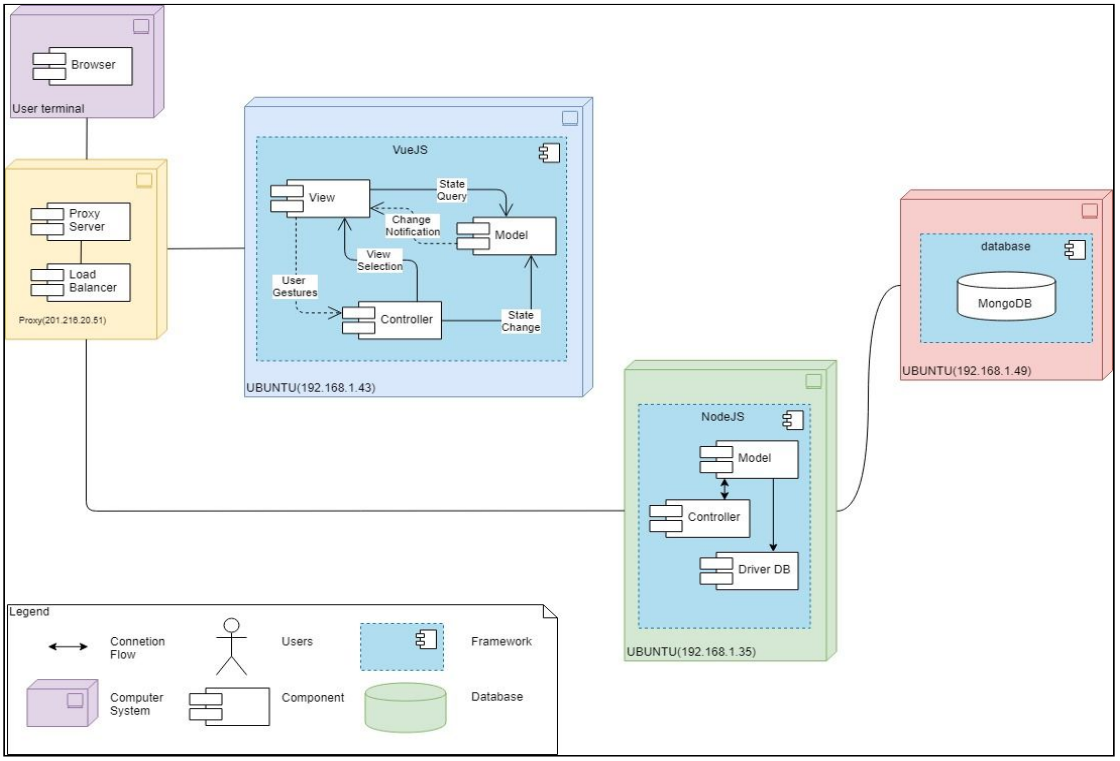
I. **Mantener múltiples copias de datos:** Para asegurar que los tiempos de respuesta de la base de datos sean aceptables, al no depender de un único repositorio de datos.
II. **Agendar uso del Recurso:** Para esto se asigna un tier Adicional que va a contener un balanceador de carga y un proxy para atender las peticiones de los usuarios y luego pasarlas al Tier Web para que sean atendidas.
III. **Introducir Concurrencia:** Para procesar las tareas relacionadas con el cambio de estado del parqueadero (ingreso y salida del vehículo) y la generación de la factura para el conductor y el operario del parqueadero.

2. Modelo de la Vista Funcional.



Es este modelo se explica en detalle como es el proceso de una petición de generar factura al sistema, Se muestra el flujo de proceso y los componentes que interactúan para poder obtener la factura.

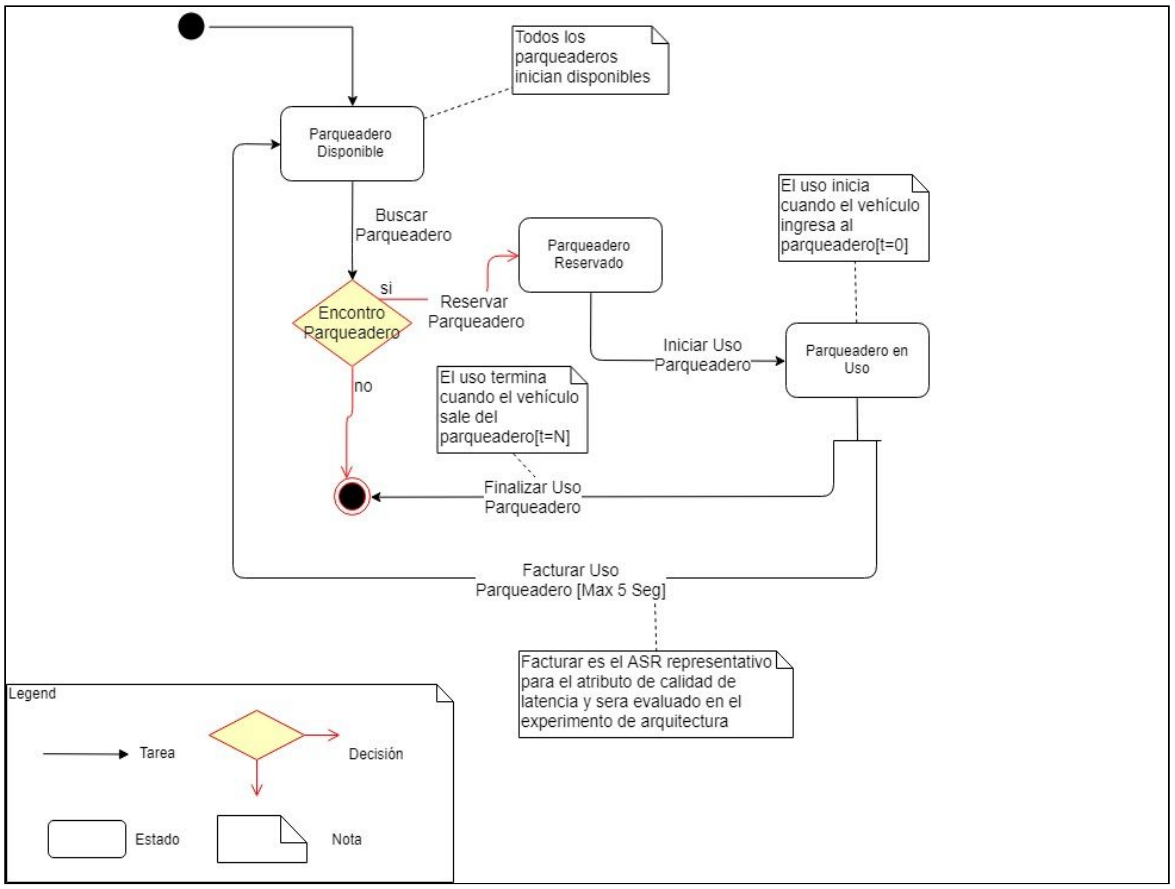
3. Modelo de la vista de despliegue



Este modelo muestra cómo va a ser la asignación de los diferentes componentes de acuerdo a la arquitectura multi-tier planteada. El HAProxy está en el tier de balanceo de carga, el application server marcado con color azul representa el tier web recibe las solicitudes del usuario y se encarga de procesar eventos y dar respuesta a las peticiones del usuario, los componentes restantes del

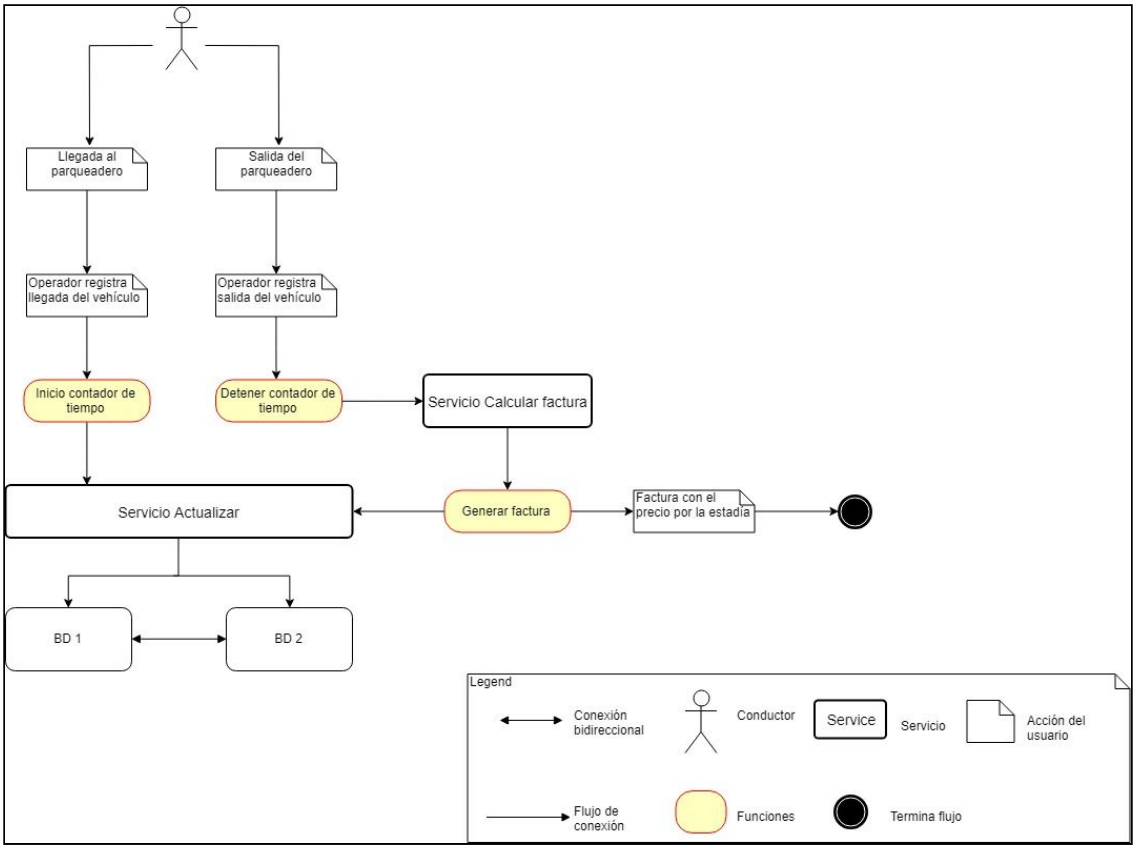
modelo son parte del tier de Backend, quién maneja la lógica del negocio e interactúa con la base de datos.

4. Modelo de la Vista de Información



En este modelo podemos ver como es el flujo de la información durante el proceso de uso del parqueadero hasta obtener la factura. En el modelo se indica el estado inicial de la información relacionada al parqueadero en especial el estado del uso de este, se indica los cambios que puede tener este atributo.

5. El grupo presenta modelos de la vista de concurrencia



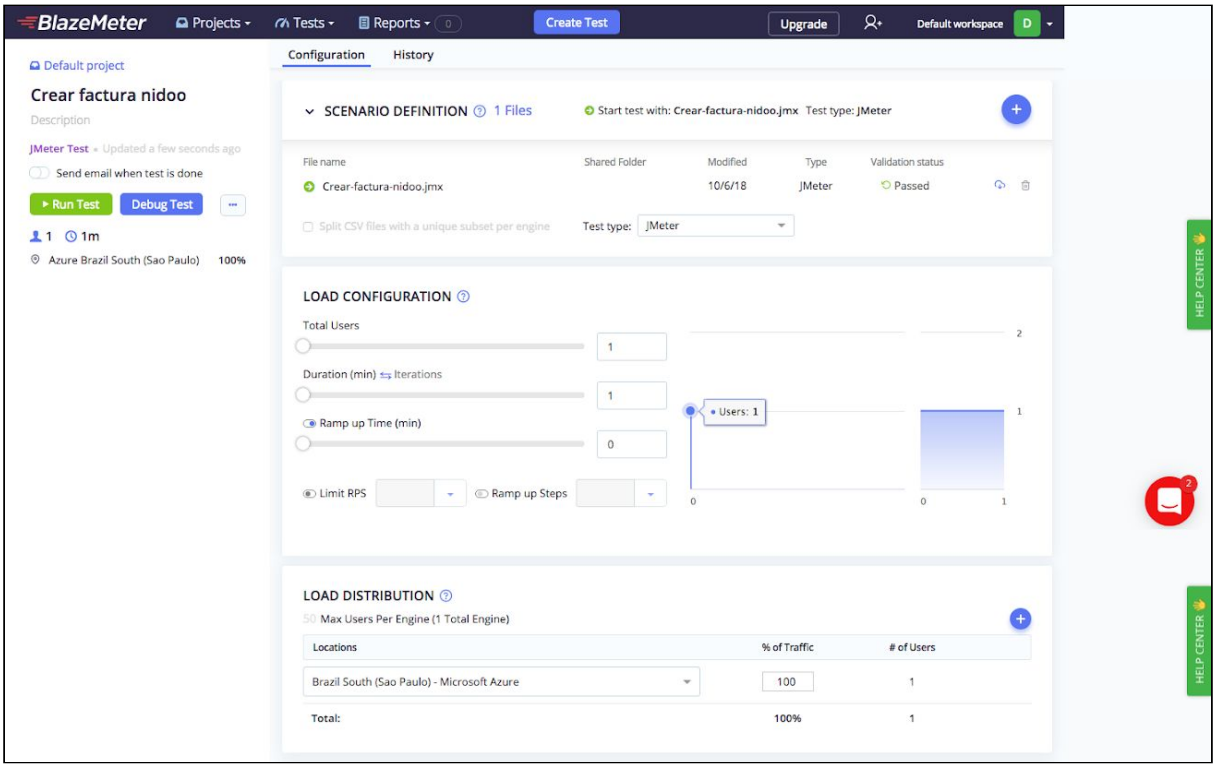
En este modelo se presenta los procesos que se pueden llevar a cabo de forma paralela, la entrada y salida de usuarios del sitio de parqueo se realizan de forma concurrente, para permitir el proceso de ingreso de un usuario al mismo tiempo que el proceso de salida de otro.

6. **Diseño del experimento que permite validar sus decisiones de diseño.**

Dado el ASR de mayor prioridad que elegimos para el experimento, se requiere calcular el pago del servicio de parqueo en menos de 5 segundos y lograr visualizar la factura. Para esto, se realizó el experimento de la funcionalidad usando Blazemeter.

El experimento consiste en ingresar a la página dispuesta para crear la factura y llenar un formulario con datos básicos (placa, nombre del cliente, cantidad de minutos y el valor por minuto); esta información será enviada para calcular el costo del servicio.

En Blazemeter se creó un nuevo proyecto y en él se creó el test para validar nuestras decisiones de diseño. Se parametriza el test de tipo Jmeter dado que ya tenemos alguna experiencia con esta herramienta; se configuró para la prueba un usuario y 1 minuto para la duración de la prueba. De las locaciones dispuestas donde están alojados los servidores de prueba se seleccionó uno alojado en Sao Paulo, Brasil. (Ver imagen)



El script de Jmeter usado para el test se encuentra en el repositorio de GitHub del proyecto:

https://github.com/MISO-4206/201820-Repo-Grupo-01/blob/master/src/latencia/Scripts%20Prueba/Crear-factura-nidoo.jmx

7. **Implementación de su experimento, describiendo los elementos utilizados en la infraestructura.**

Se utilizaron 1 maquina fisica, 2 máquinas virtuales conectadas en una red Ethernet/1000

Dirección IP MV	Descripción	Configuración
201.216.20.51 ip publica	Proxy/LoadBalancer	Router corriendo firmware open source https://en.wikipedia.org/wiki/Tomato_(firmware)
192.168.1.43	Front-End	Ubuntu Desktop 18.04, 8GB RAM, MV2 265GB.
192.168.1.35	Back-End	VM Ubuntu Desktop 18.04, 2GB RAM, SSD 10GB.
192.168.1.49	Data-Base	VM Ubuntu Desktop 18.04, 2GB RAM, SSD 10GB.

Se solicitaron máquinas virtuales pero se presentaron varios inconvenientes de configuración, que se describen más adelante en el presente documento. Por lo que se decidió utilizar nuestra propia infraestructura.

8. Descripción las herramientas utilizadas para su experimentación y el propósito de estas.

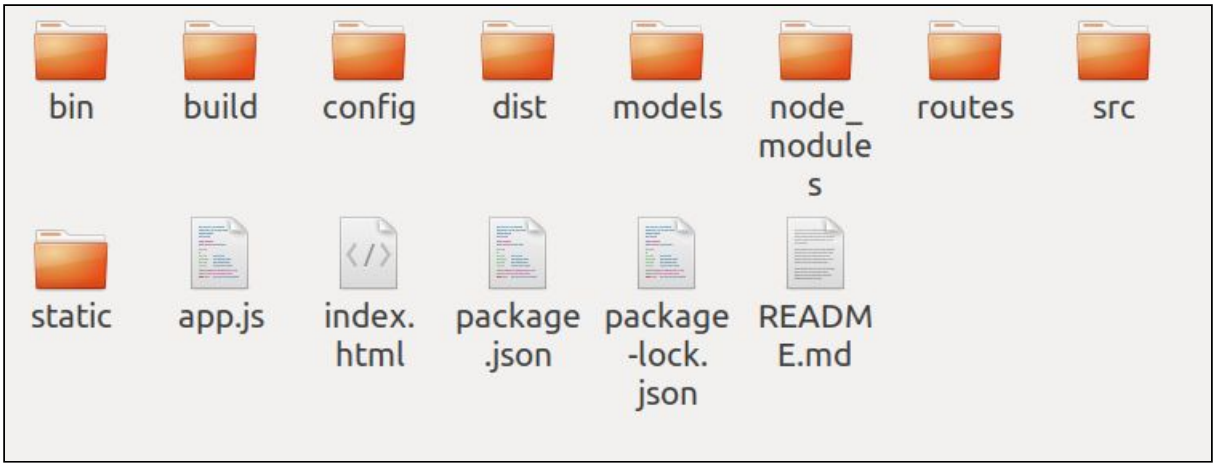
Dirección IP MV	Descripción	Uso Componente
201.216.20.51 ip publica	Proxy/LoadBalancer	Puerta de acceso a la sistema de información. Balacea todo el tráfico recibido por el puerto 3000 con el algoritmo Round-Robin y lo redirige al nodo 122 en donde se encuentra el Front-End de la aplicación. https://en.wikipedia.org/wiki/Tomato_(firmware)
192.168.1.43	Front End	Aplicación en Vue.js que se encarga de crear una factura y listar el total de facturas.
192.168.1.35	Back-End	Expone el CRUD de la factura mediante servicios REST hacia el back-end
192.168.1.49	Data-Base	Base de Datos No relacional MongoBD encargada de almacenar los documentos en formato JSON

La aplicación se implementó bajo el Stack MEVN (Mongodb, Express, Vue.js y Node.js), este stack permite crear aplicaciones web en lenguaje javascript que se ejecutan del lado del servidor. Se implementó el ASR de generar la factura, pues según la priorización realizada por el grupo se determinó que este es el más importante para el atributo de calidad de Latencia.

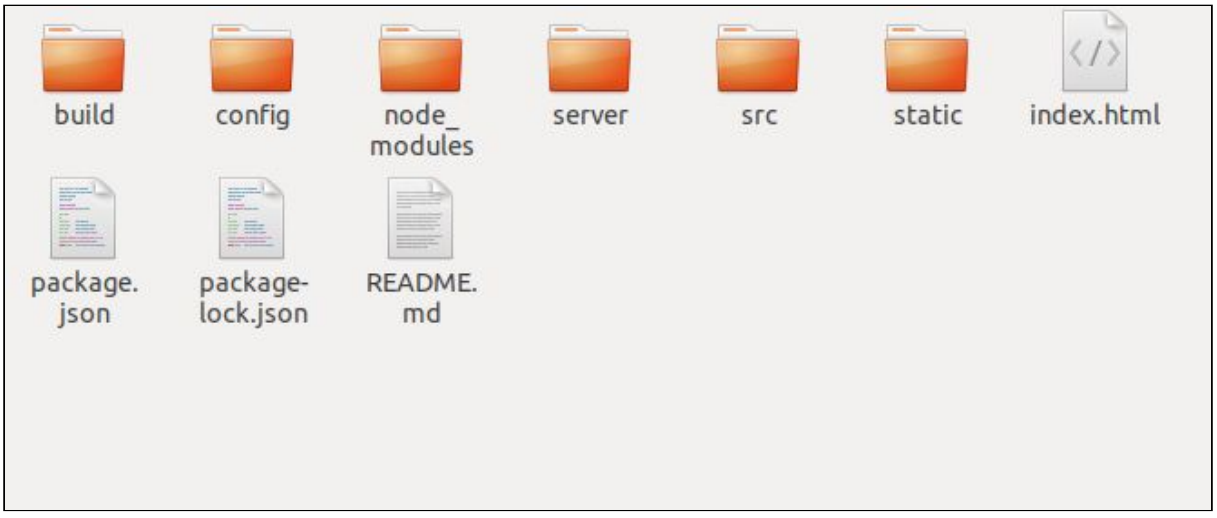
Los formularios construidos permiten mostrar el total de facturas previamente generadas, además permite crear una nueva factura y consultar una factura en el modelo On-Demand para el sistema Nidoo.

Esta aplicación es una Single Page Application utilizando Vue.js, permitiendo una interacción fluida con el cliente ya que no carga en segundo plano los recursos que necesita en cada operación. Esta estará alojada en un servidor Node.js que utiliza el motor v8 de javascript de Google para ejecutarlo del lado del servidor, además se usa Express como framework del lado servidor y Mongodb como Base de datos no relacional para almacenar la información; en el caso de este experimento se almacena una factura en formato JSON.

Estructura del Proyecto en Front-End:



Estructura del Proyecto en el Back-End:



Vista prototipo

http://casabogota.mynetgear.com:3000/#/

9. **El grupo presenta los resultados y realiza un análisis de ellos, describiendo qué acciones de mejora deberían ser tomadas en la arquitectura, si es el caso.**

La prueba se realizó usando el complemento de Blazemeter para Google Chrome que permite previamente grabar los pasos que se realizarán durante el test para la creación de la factura. Posteriormente se realizó el test a la aplicación y se reflejó la creación de una factura con un usuario. Los resultados del test se presentan a continuación:

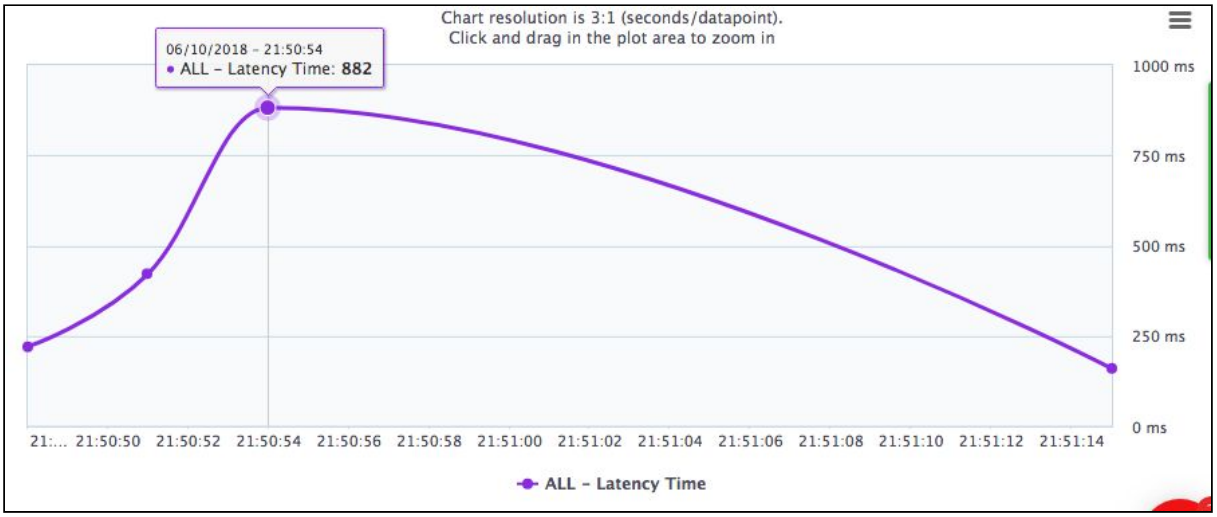


Figura 1: Latencia de la prueba realizada con Blazemeter durante el tiempo de la prueba.

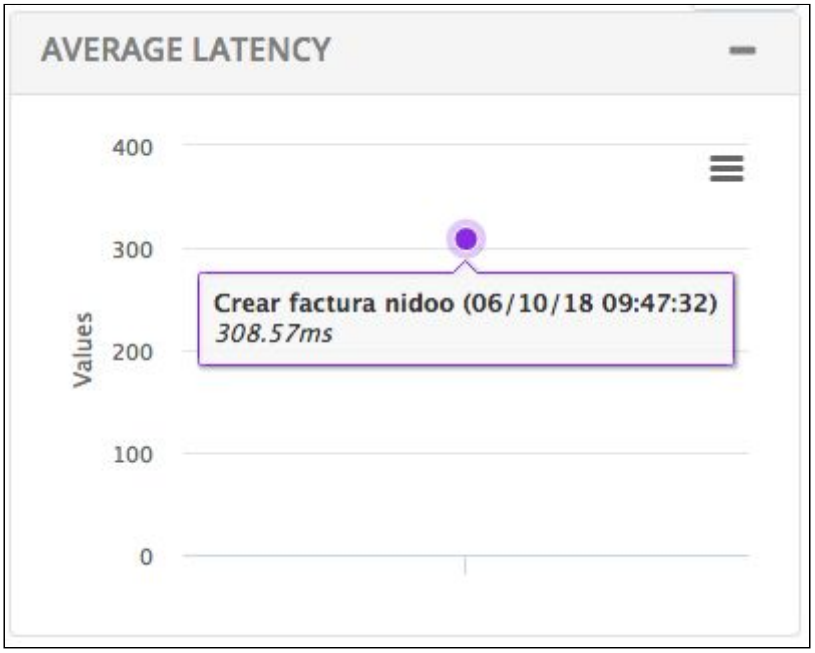


Figura 2: Promedio de latencia de la prueba realizada con Blazemeter.

El test arrojó que la creación de factura tuvo un pico de 882ms y la latencia promedio del test fue de 308.57ms, esto refleja que la arquitectura propuesta y las herramientas utilizadas para el desarrollo de la funcionalidad soportan la métrica de 5 segundos que se propuso en el ASR.

Este resultado nos deja ver que la latencia de la aplicación está muy por debajo de la métrica, lo que nos da el cumplimiento no solo de esta ASR sino que nos brinda un margen amplio para cumplir con otros requerimientos de calidad.

Punto de sensibilidad identificado

Llegamos a la conclusión que el componente proxy definido en la arquitectura es determinante, ya que permite seleccionar, priorizar y enlazar las capas intermedias de los diferentes componentes, ya que esta capa agenda el uso de los recursos y separa la capa web.

10. El grupo presenta los aspectos que han funcionado bien en el equipo

Se resalta que en el grupo se realizan revisiones cruzadas de los modelos y el código fuente realizado, esto facilita que todos los integrantes del equipo tengan el mismo entendimiento del problema y las soluciones propuestas.

Las tareas durante este sprint se organizaron y se asignaron de forma que todos los integrantes siempre tenían una labor asignada, se destaca el compromiso de los integrantes del grupo para cumplir con las asignaciones realizadas.

Durante el desarrollo del sprint se realizaron varias sesiones de trabajo para validar el avance y comentar dificultades encontradas.

El grupo mantiene una buena comunicación, lo que permite que el trabajo durante las sesiones virtuales sea fluido y todos lo integrantes aporten al proceso de desarrollo y documentación del trabajo realizado.

11. **Se presentan los aspectos que no han funcionado bien en el equipo y como se corregirán para próximas entregas.**

Haciendo uso de la capacidad de cómputo de la universidad, se nos entregaron 4 máquinas virtuales con las Ips: 172.24.100.124/157.253.238.35, 172.24.100.123, 172.24.100.122 y 172.24.100.121

Se configuró inicialmente la arquitectura planteada hasta tal punto que se presentaron dificultades en la configuración de red y de algunos componentes para enlazar la capa de frontEnd y BackEnd con el Balanceador.

Por tal motivo, el grupo optó por configurar la arquitectura de manera remota en los equipo de cómputo con lo que cuenta el grupo de trabajo fuera de la arquitectura de la universidad, obteniendo control total de la configuración de los componentes y logrando implementar, realizar las pruebas y verificar los resultados.

Es importante resaltar que la falta experiencia en el uso de las tecnologías utilizadas tales como: VueJS, nodeJS, mongoDB, haProxy; provocó dificultades para implementar más rápido la arquitectura planteada.

La disponibilidad para reunirnos como grupo se vió afectada nuevamente por compromisos en otras clases.