**Informe Experimento 1 SoluMovil.**

**Pre experimentación**

**Problemática**

La ubicación y el seguimiento de un gran número de vehículos es un aspecto de vital importancia para la empresa de transporte Tbc. Por esto Tbc ha decidido contratar a SoluMovil para diseñar y construir una aplicación que sea capaz de responder ante un gran número de solicitudes simultáneas de sus vehículos (Mobibus, vcub y tranvía eléctrico) así como una respuesta rápida ante estas. Adicionalmente está el almacenamiento de los datos generador por lo vehículos así como el diseño e implementación de aplicaciones que permitan a estos comunicarse con el servidor principal.

**Objetivo del experimento**

Aceptar, procesar y responder a las solicitudes de múltiples usuarios simultáneos en la aplicación SoluMovil con el fin de satisfacer las necesidades de la empresa Tbc . Esto, a través de la implementación de aplicaciones StandAlone capaces de comunicarse con un servidor.

**Descripción del experimento**

Para cumplir con los requerimientos del cliente se siguieron los siguientes pasos en orden.

1. Comprensión global de la problemática, abstracción del problema a UML con casos de uso para latencia y Escalabilidad.
2. Investigación de arquitecturas, pros y contras en cuento a latencia y tiempo de respuesta.
3. Construcción del software capaz de responder a 4500 solicitudes en menos 5 segundos o menos manteniendo una latencia máxima de 1 segundo.
4. Pruebas con las arquitecturas seleccionadas para el tiempo de respuesta y la latencia de los requerimientos.
5. Implementación de la base datos.
6. Realización de las aplicaciones StandAlone.

**Artefactos a construir**

Cómo el cliente solicito se construyeron tres aplicaciones stand alone para Mobibus, vcub y tranvía eléctrico. Así como una aplicación servidor capaz de comunicarse y responder a las Standalone.

**Recursos de experimentación**

Seleccionar los recursos no fue una tarea sencilla. Se buscó una arquitectura capaz de soportar la escalabilidad y la latencia. Así como herramientas que permitiesen el desarrollo más fácil y efectivo.

**Arquitectura a utilizar**

La arquitectura escogida para el diseño de la aplicación SoluMovil es MTV: *model*, *template* y *view*. En esta arquitectura *view*  describe qué datos están representados, el *template* cómo están representados estos datos y *model* las operaciones hechas con los datos. Pero, ¿Dónde está el *controller*? En esta arquitectura es la maquina en sí misma. (Django, s.f.). EN la imagen 1 se muestra un representación gráfica de la arquitectura.

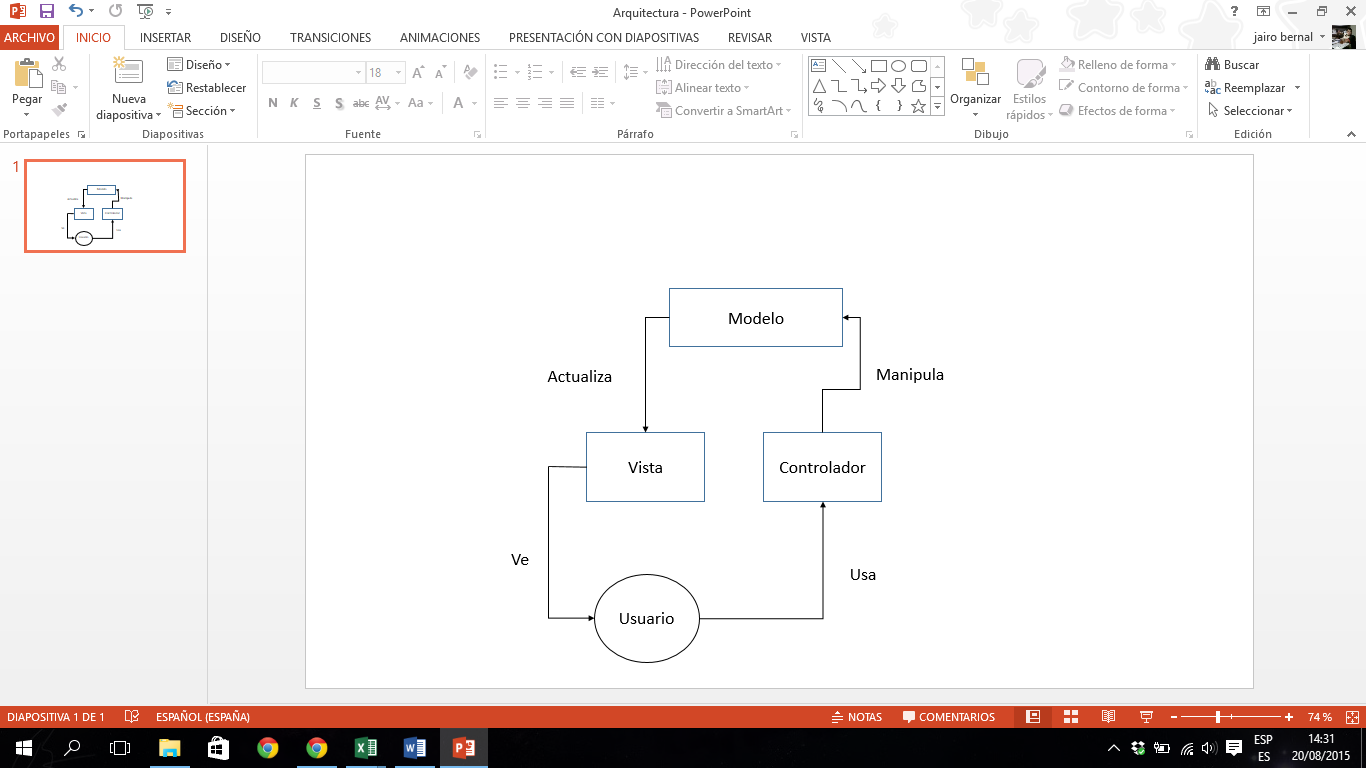


Imagen 1 Arquitectura seleccionada para la aplicación

**Lenguaje de programación**

Se escogió Python ya que este lenguaje de alto nivel permite realizar instrucciones de manera sencilla y concisa. Además los Frameworks, servidores y librerías a utilizar están basados en este lenguaje.

**Framework**

Django ya que este facilita el proceso de desarrollo, tanto en la parte web como en la parte de persistencia.

**Servidor**

Se probaron dos servidores uWSGI un poderoso servidor basado en “piezas” que permite correr diferentes plataformas y lenguajes permitiendo así un crecimiento a futuro (uWSGI , s.f.). El segundo es gunicorn que está basado en Python y facilita su integración con Django. (Gunicorn , s.f.)

**Pruebas**

Jmeter Esta librería de java permite probar la latencia y escalabilidad de la aplicación de forma fácil y segura.

**Base de datos:**

El estilo de base de datos utilizado es *POSTGRES SQL* junto con Django. Esto, ya que Django permite la programación en *POSTGRES SQL* sencilla y efectivamente. Adicionalmente, la base de datos es local.

**Otras librerías**

**GeoDjango**

Potente paquete que permite utilizar google y modelos matemáticos para calcular rutas a partir de datos de gps así como el seguimiento y la representación. (Django, s.f.)

**Hardware**

Los servidores son compatibles con una gran variedad de sistemas operativos exceptuando a Windows por ende el hardware utilizado es Mac, en particular *MacbookPro* con ocho *Gigabytes* de memoria RAM y cuatro *cores*.

**Resultados esperados**

Se espera que al final del experimento métodos tanto POST como GET puedan soportar 4500 usuarios o más en 5 segundos o menos teniendo una latencia menor a un segundo. Adicionalmente, que los datos se guarden en la base de datos y que aplicaciones Stanadlone puedan comunicarse con la aplicación servidor.

**Duración y etapas**

En la tabla 1 se puede ver la planeación del proyecto así como las etapas en las que se dividió el proyecto.

Tabla 1 Tiempos Planeados de desarrollo

|  |  |  |
| --- | --- | --- |
|  |  | Tiempo estimado (Horas) |
| Planeación | Requerimientos de las aplicaciones StandAlone | 1 |
| Información que el Servidor debe recibir | 1 |
| Protocolo | 1 |
| Desarrollo | Mobibus StandAlone | 5 |
| Vcub StandAlone | 5 |
| Tren Eléctrico StandAlone | 5 |
| Desarrollo Método Get | 2 |
| Base de datos | 1 |
| Desarrollo Método Post | 2 |
| Implementación comunicación | 10 |
| Pruebas | Pruebas | 5 |
| Corregimiento | 5 |
| Pruebas | 5 |
| Otros | Instalación software | 3 |
| Desarrollo reporte | 1 |
|  | Total Estimado | 208 |
|  |

En la planeación se puede observar que la planeación y el desarrollo son etapas clave, mientras que en las pruebas se turan el probar y corregir par a obtener los mejores resultados. La razón por la cual las tareas tienen tan bajo tiempo estimado es porque en el experimento pasado se realizaron las aplicaciones StandAlone así como la implantación de la base de datos.

**Post Experimentación**

**Método GET:**

En la fase de pruebas se experimentó con destinos servidores en destinos computadores, los primeros resultados no fueron satisfactorios. Por ejemplo la imagen 1, muestra una desviación estándar y un porcentaje de error altos. Esta prueba fue ejecutada con un Mac de 4GB utilizando Uswgin. Otro problema fue el de la replicación de resultados, al correr las pruebas en un momento y luego en otro los resultados variaban mucho como se muestra en las imágenes. Se atribuye esto al hecho de que en la universidad se conectan muchas personas a diferentes horas del día a la red. Haciendo que la latencia de la aplicación aumente por causas eternas. Este problema puede ser reducido al conectar las maquinas, las de prueba y las de ejecución, con un cable *Ethernet*.

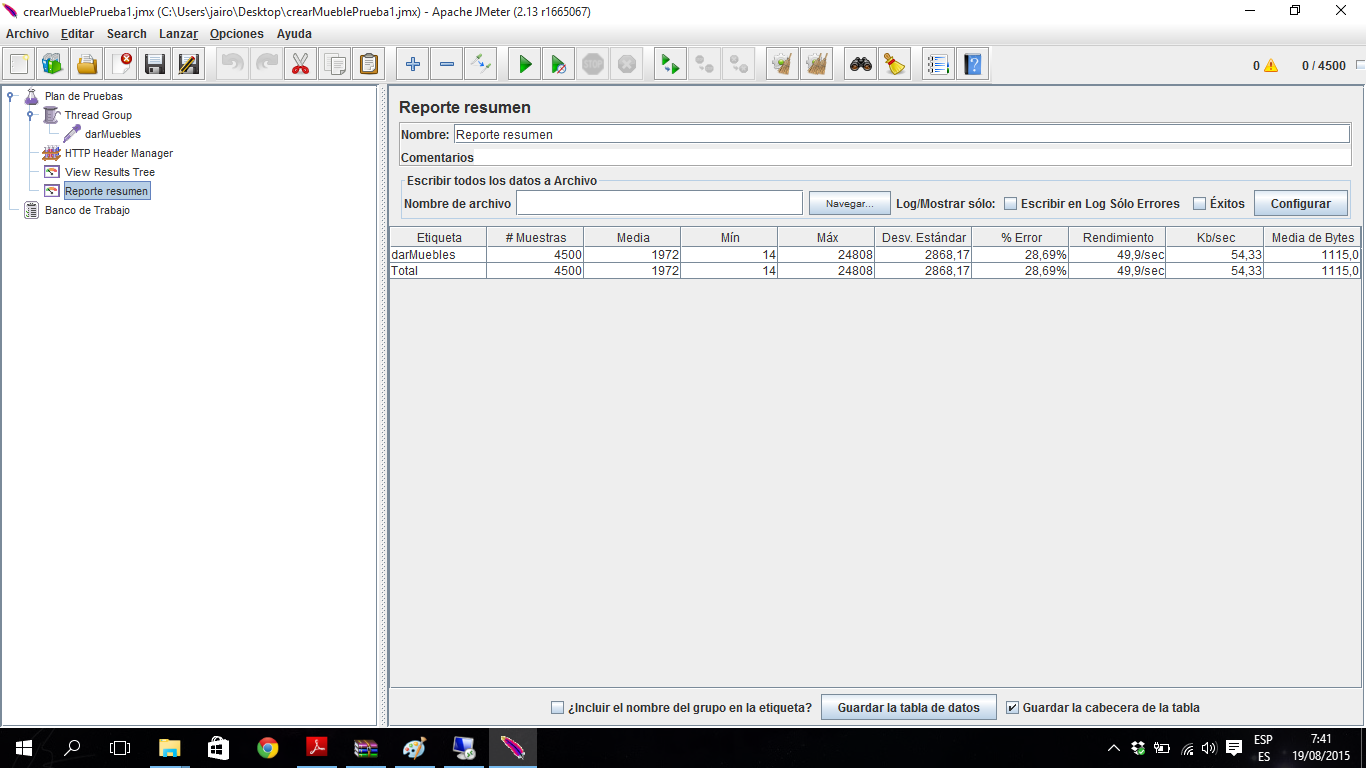


Imagen 2 Resultados mostrados por JMeter en la primera fase de pruebas

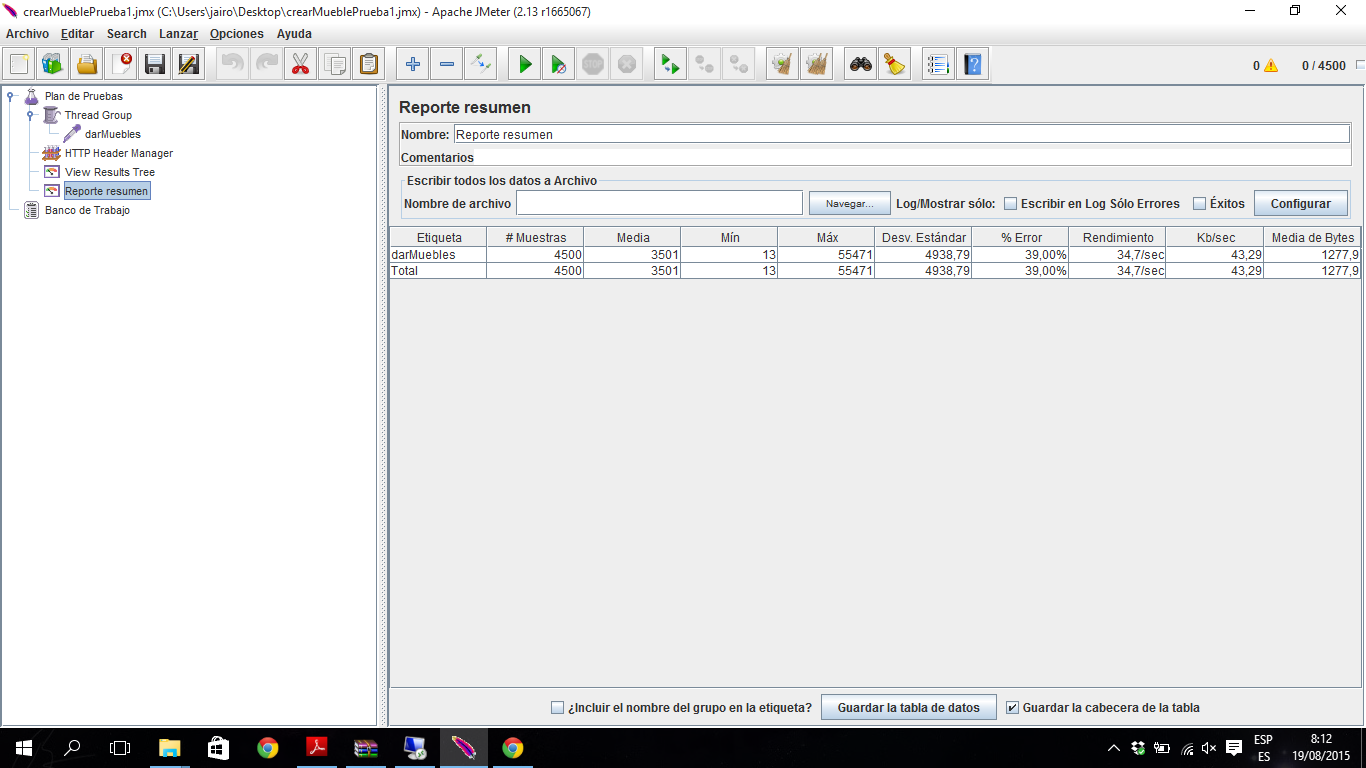
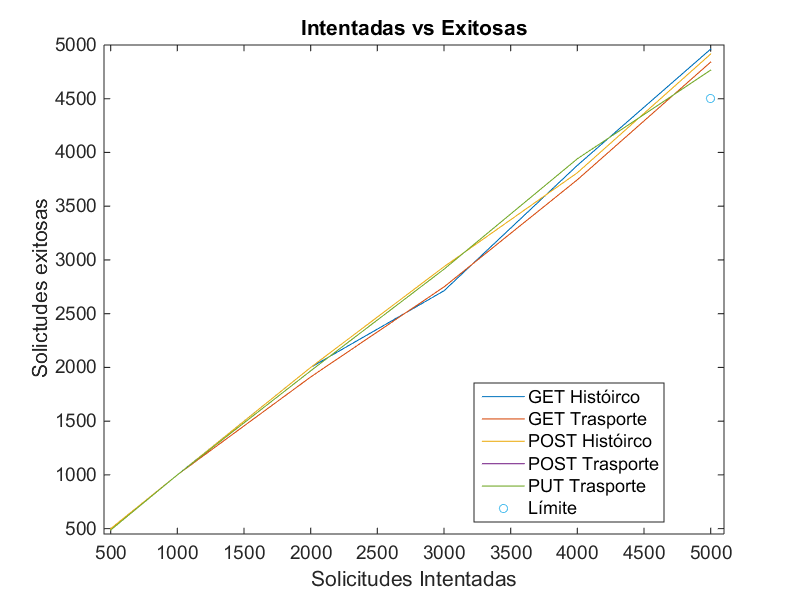


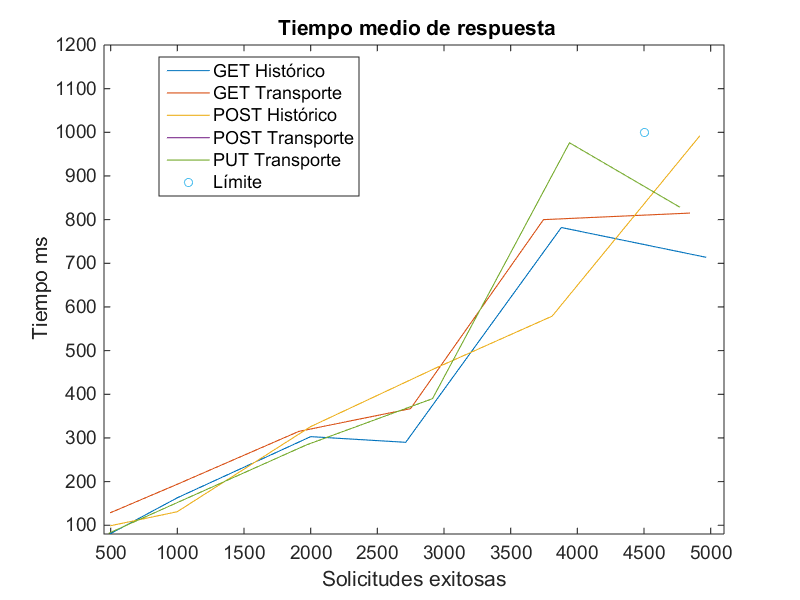
Imagen 3 Resultados mostrados por JMeter en la primera fase de pruebas

En la segunda etapa de desarrollo se cambió la configuración por defecto de GUnicorn. Esto permitió disminuir el tiempo de respuesta, sin embargo el error no disminuyo a cero. En la gráfica 1 se observa las solitudes enviadas al servidor contra las solicitudes que se respondieron exitosamente.



Gráfica 1Solicitudes intentadas vs Solicitudes Exitosas

En la gráfica 2 se decidió tomar el número de solicitudes exitosas vs el tiempo medio de respuesta, esto ya que ese fue el número real de solicitudes que se respondieron.



Gráfica 2 Tiempo medio de respuesta

**Resultados**

La implementación de la base de datos se hizo desde el taller pasado y por ende los resultados no se vieron afectados.

**Tiempos reales**

Los tiempos reales se pueden ver en la tabla 2

Tabla 2 Tiempos reales de desarrollo

La mayoría de este experimento se hizo en el pasado. Por esto

**Artefactos construidos**

El servidor principal fue construido, sin embargo las especificaciones del cliente de latencia y escalabilidad solo se cumplen para los métodos GET y POST se cumplieron los atributos de calidad estipulados. No se hizo DELETE ya que el sistema no debe eliminarlos datos existentes, Solo registrar nuevos, leerlos y modificarlos.

Se implementa la base de datos local utilizando postgreSQL. Así mismo la interfaz de usuario utilizando Django y

**Análisis**

Se decidió utilizar PostgresSQL ya que su implementación con Django fue fácil. Además, permite modelar los datos lógicamente, donde es su mayoría las tablas, llaves e IDs se crean automáticamente. Sin embargo, algunos ids son cambiados para obtener mayor velocidad en la consulta de la base de datos.

**Conclusiones**

En aspectos generales este primer experimento fue exitoso, se aprendió utilizar nuevas herramientas, diseñar arquitecturas y modificar parámetros de software para obtener resultados deseados. Se cumplió con el requerimiento de escalabilidad pedido para POST, GET y PUT mientras que el de latencia solo para GET y POST. Adicionalmente el grupo entendió la importancia del trabajo en equipo y la comunicación entre los miembros así como la importancia de tener un ambiente de trabajo cómodo y productivo.