**JSOFT – Experimento 1 (Entrega Definitiva)**

Johan Velázquez 201115956/ Karen Osorio 201126600 / Santiago Robayo 201216389/ Juan Sebastián Arciniegas 201325828

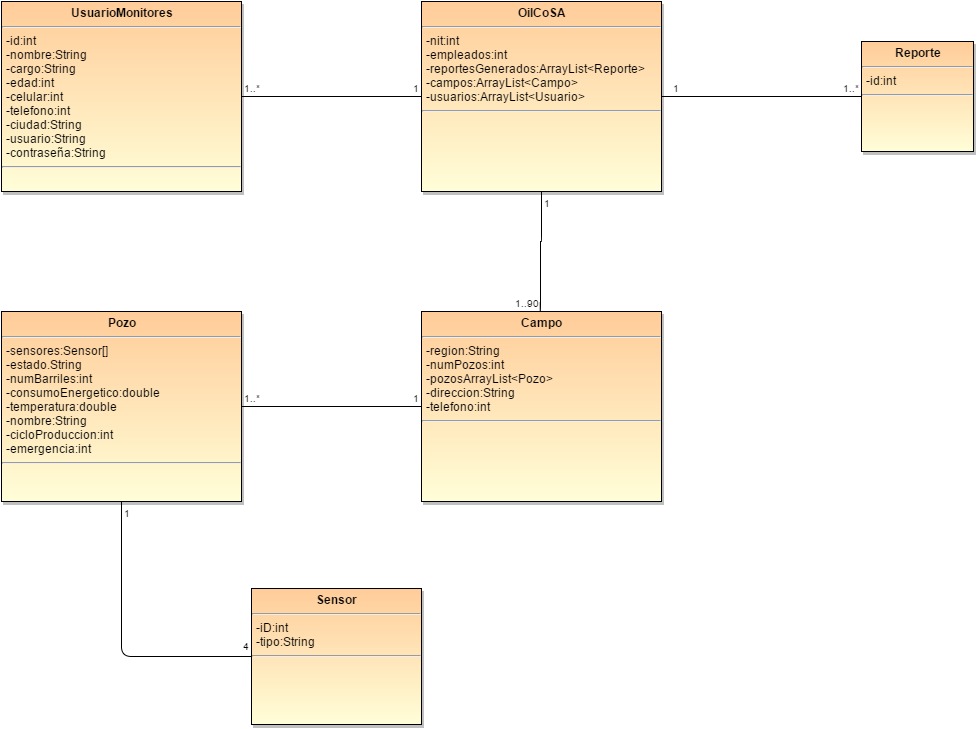
**Justificación Arquitectura:**

El proyecto será basado en servicios REST de JAX-RS. Ya que debido a la facilidad de manejo que esta otorga y por el potencial de desarrollo de manera eficaz y ágil. Adicionalmente esta arquitectura cuenta con los beneficios de separación por capas lo cual permite un fácil manejo de sus distintos componentes, facilitando así su mantenimiento en el caso de que se deseara una aplicación a tiempo indefinido y además su fácil modificación en caso de ser requerida.

Además, el cumplimiento de los atributos de calidad de desempeño y escalabilidad. Se ve que una aplicación REST soportaba de manera eficaz tanto el desempeño como la escalabilidad para un número significante de solicitudes. Es decir que JAX-RS permite manejar de manera adecuada el número de solicitudes que se espera para la aplicación realizada durante el experimento de 4800 sensores.

**Pre-experimentación:**

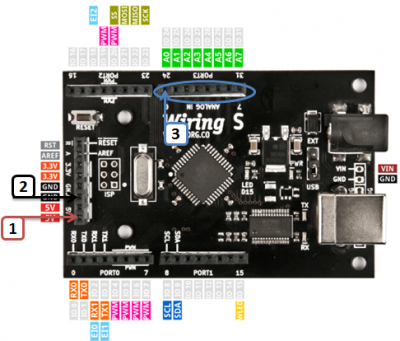
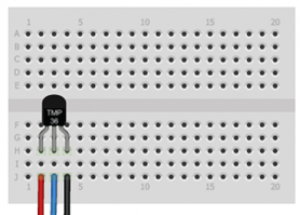
* **Problemática:** Se plantea desarrollar una arquitectura por capas que use servicios REST para la comunicación con los sensores por medio de peticiones HTTP y que sea capaz de cumplir con requerimientos funcionales y no funcionales de la aplicación.
* **Objetivo del experimento:** Probar la arquitectura planteada para verificar el cumplimiento de los requerimientos no funcionales
* **Descripción del experimento:** Se desarrollará la capa de persistencia, la capa de presentación y se expondrán nuevos servicios con el fin de suplir otros requerimientos funcionales. Por medio de pruebas de carga se probará la latencia y escalabilidad del software.
* **Artefactos a construir:** Se construirá la capa de persistencia dentro del backend del proyecto y en otra capa se realizará la interfaz gráfica para que desde allí se consuman los recursos de la aplicación.
* **Recursos de la experimentación:** El experimento se desarrollará en el ambiente de ejecución NetBeans 8.1, como servidor para desplegar la aplicación se utilizará GlassFish 4.1.1 y para las pruebas de carga se utilizará JMeter.
* **Resultados esperados:** Se espera que el sistema pueda atender las 4800 solicitudes que requiere los pozos en menos de un minuto, además que en caso de una emergencia actualizar la información.
* **Duración y etapas:** Esta etapa finalizará para el 31 de agosto de 2016. Deberá incluir pruebas de carga de la arquitectura planteada y la incorporación de la persistencia. Se realizará entregas semanales de avances progresivos. Inicialmente se llevará a cabo la incorporación de la persistencia.

****

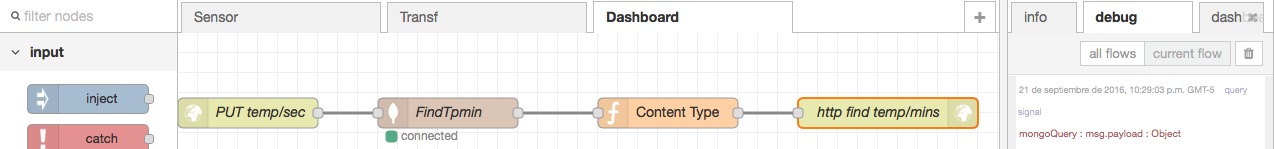
**Actualizaciones Pertinentes:**

**Toma de datos**

Por medio de un sencillo circuito de un sensor de temperatura (LM35DZ) y un microcontrolador Wiring S, el cual se puede ver en las próximas imágenes, se puede tomar la temperatura de los pozos petroleros.



Para lograr tomar la temperatura se utilizó un flujo de datos a través del framework node-red. Gracias a node-red la información recogida por el sensor es enviada a través de un JSON por correo el electrónico el cual informa la temperatura actual del pozo.

****

****

En dado caso que esta temperatura supere los 27 C se manda un aviso JSON de emergencia al correo electrónico que avisa la alerta encontrada.

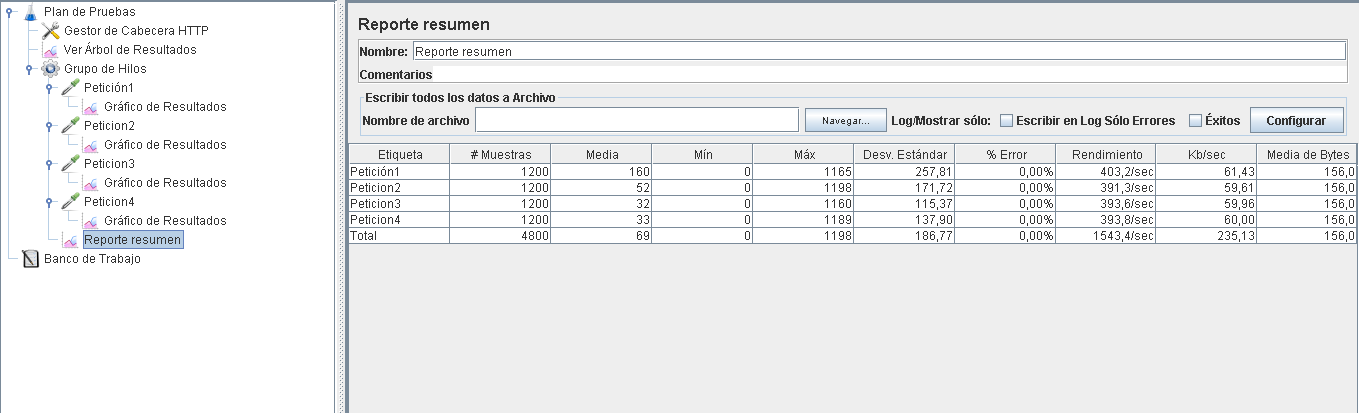
****Para almacenar y persistir los datos se está utilizando nuestro propio servicio de Derby la cual recibe los JSON por medio de peticiones http (PUT).

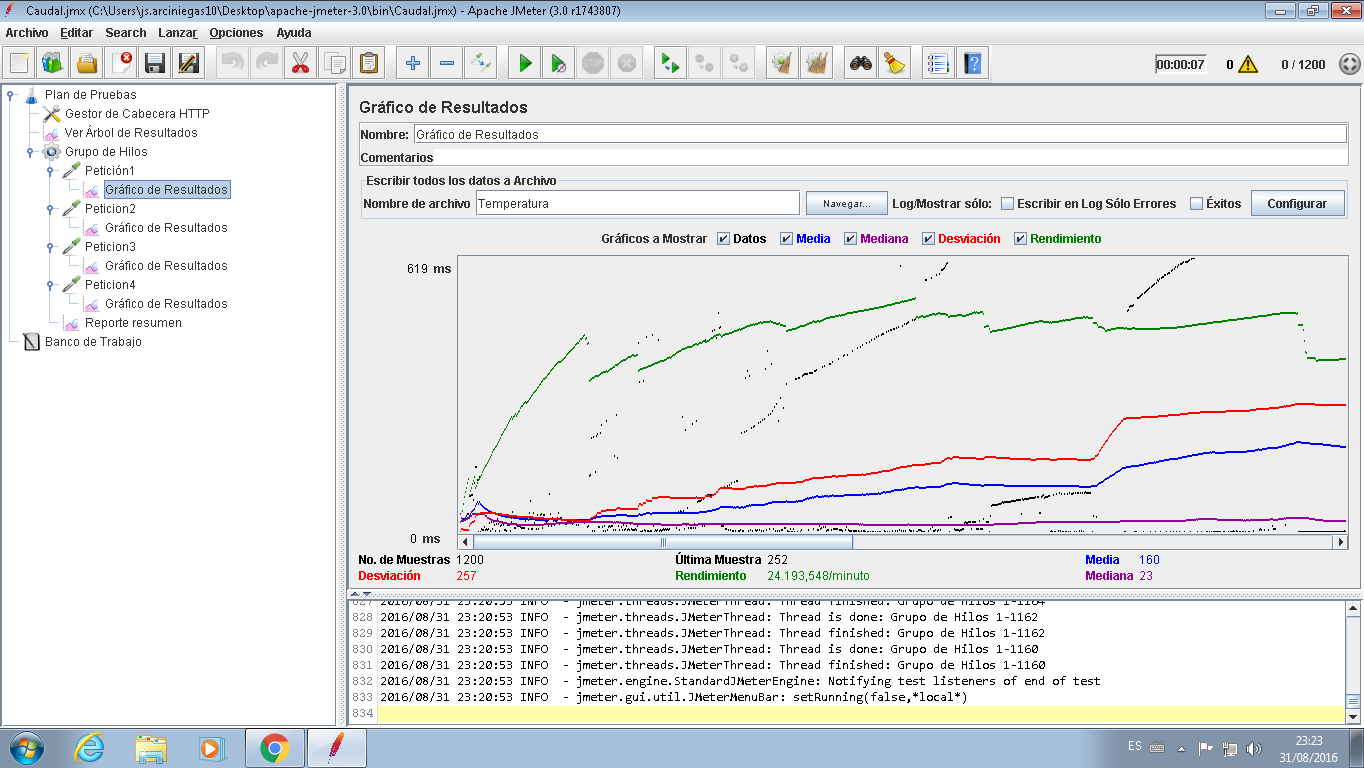
**Experimentación & Comparación nuevos datos:**

Para este experimento se desarrolló y ejecutó las pruebas de carga de los siguientes requerimientos:

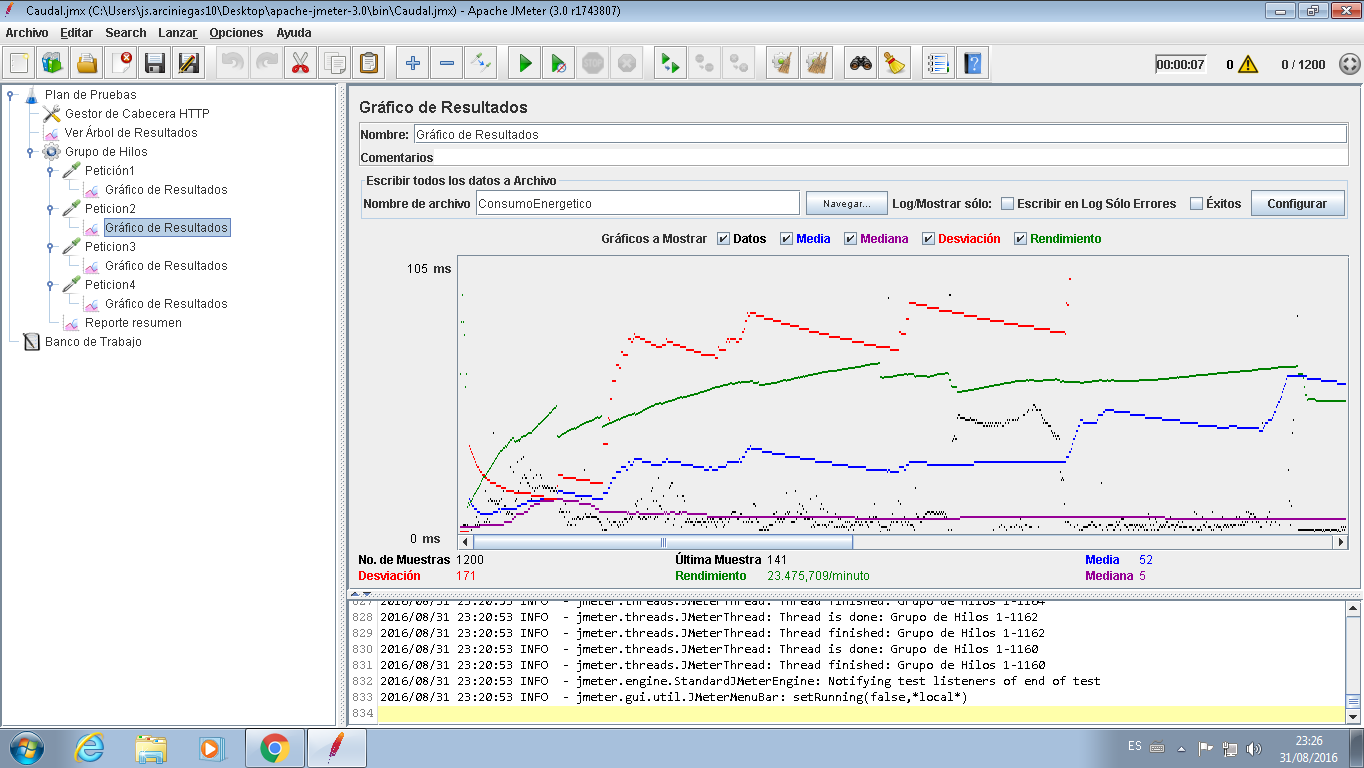
* Generación de reporte de carga en los 4800 sensores en 1 segundo
* La actualización de la información de cada sensor en (caudal diario, consumo diario de energía, temperatura, ocurrencia de emergencia)
* La actualización de la información de 4800 sensores (Escalabilidad)

A continuación, se muestran algunos resultados obtenidos para estas pruebas y sus respectivos análisis:

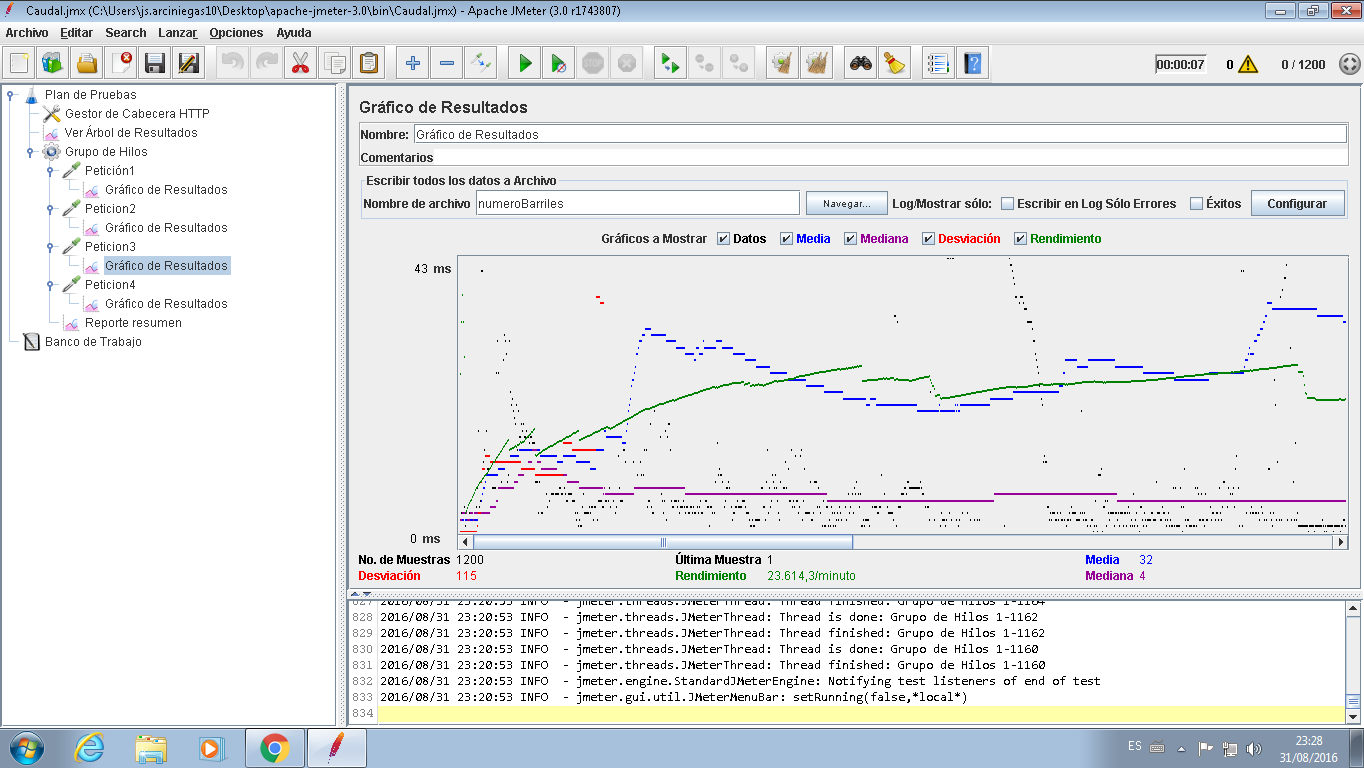


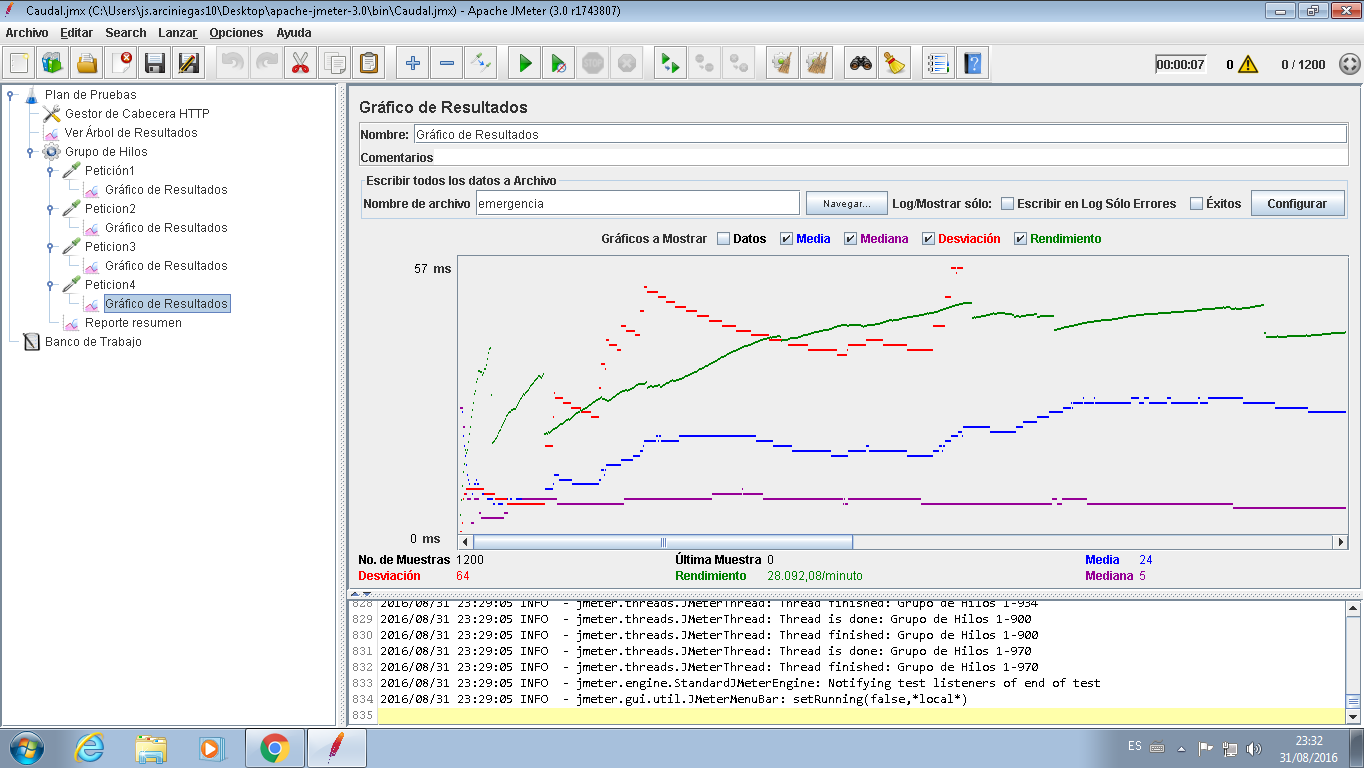


En la primera gráfica se puede apreciar que en la prueba de latencia para que generar un reporte de temperatura, con 1200, se obtiene una media de 160 ms de respuesta por cada ThreadGroup. Lo que garantiza que latencia en la generación de un reporte es baja. Así mismo, la segunda gráfica lo constata lo mismo con el consumo energético.



Se observa en las gráficas que tenemos un rendimiento de 23-28 threads por minuto.





**Post-experimentación:**

* **Resultados obtenidos:** Se obtuvieron resultados positivos para las 4 pruebas de latencia. Con un porcentaje de error del 0% aun que si bien el rendimiento por carga no es el mejor se cumple la actualización de carga cada segundo.
* **Duración real:** A pesar que se cumplió con los tiempos establecidos, los resultados obtenidos no son del todo satisfactorios ya que se presentan fallas de rendimiento en
* **Artefactos construidos:** Se desarrollaron todos los artefactos mencionados en la pre-experimentación.
* **Análisis:** Problemas ocurridos se deben debido una petición tiene que viajar por las diferentes capas de la aplicación y además de esto una petición de inserción es más costosa que una de consulta. En segundo lugar, se les atribuye a los recursos computacionales con los que se cuenta.
* **Conclusiones:** Dado que la capa de persistencia ya estaba desarrollada desde la entrega parcial 1, concluimos también que debemos tener presente en todo momento que al implementar el envió de datos por medio del microcontrolador la actualización de las transacciones en la base de datos puede llevar más tiempo, la comparación frente al tiempo de respuesta con una petición fue casi la misma lo cual deja unos resultados satisfactorios al equipo de trabajo.