**Entrega definitiva Exp. 1**

**Integrantes:**

Juan Pablo Gonzalez P. 201424703

Felipe Martínez P. 201328315

Luis Mesa F. 201314683

Joan David Torres P. 201315711

**Introducción:**

En esta entrega se podrá observar la presentación del software OilCol con todos los requerimientos y objetivos que se pedían. En primera instancia se explica como funciona la persistencia y guardado de información de forma elocuente respecto a la arquitectura implementada. Posteriormente se comparan las pruebas de carga respecto a la entrega anterior para poder mostrar las mejoras en tiempos de las mismas (hechas en JMeter). Finalmente se expone y muestra cómo el sistema es capaz de recibir y utilizar información proveniente de sensores por medio del uso de microcontroladores.

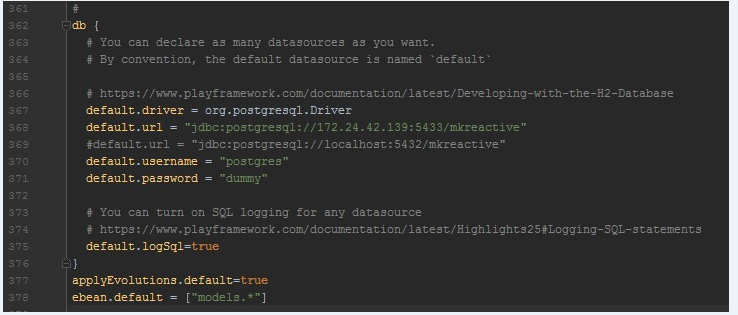
**Índice:**

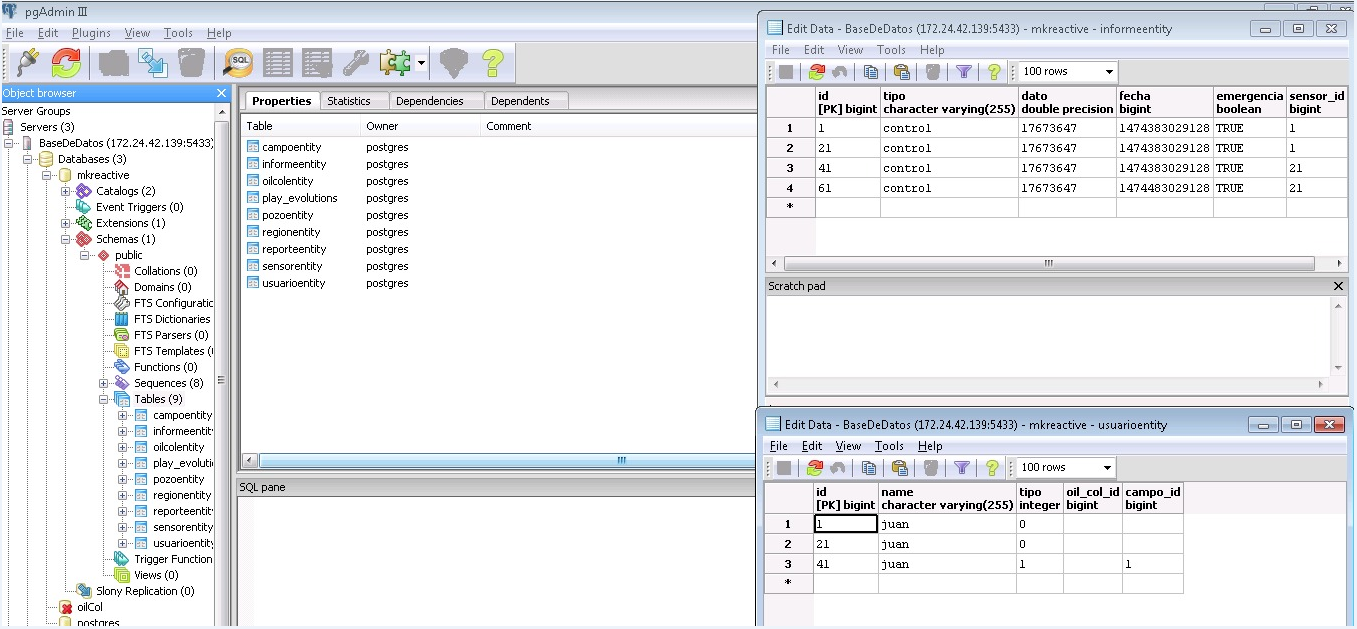
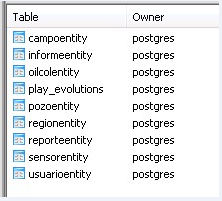
1. Desarrollar la capa de persistencia.
2. Volver a ejecutar las pruebas de carga.
3. Actualizar el documento de experimentación.
4. Analizar y comparar los resultados obtenidos en ésta entrega contra los resultados obtenidos en la entrega previa.
5. Desarrollar el programa que permite el envío de los datos (e.g,. temperatura de la bomba del pozo) desde un microcontrolador donde están conectados los sensores.
6. La lógica de negocio ofrecida por el sistema debe ser consumida por el microcontrolador.

**Cuerpo:**

1. Desarrollo de capa de persistencia:

La capa de persistencia se había hecho desde la entrega pasada. Para lograr esto la información se guarda en una base de datos utilizando Postgresql. De esta forma, los servicios guardaban y pedian informacion, esto se puede hacer desde cualquier computador de los integrantes ya que la configuración hecha en play permite la coneccion a la misma. La imágenes que se ven a continuación muestran la configuración de concepción del proyecto y la base de datos de Postgresql.





2. Ejecución de Pruebas, Análisis y comparación de las pruebas de la entrega anterior.

Recursos de la experimentación

Las máquinas siguen las siguientes configuraciones:

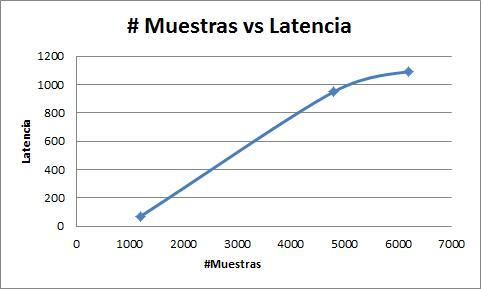
* Disponer de Java Development Kit versión 8 (JDK 8).
* Tener la configuración de la variable JAVA\_HOME en las variables de entorno.
* Disponer de un entorno de desarrollo integrado (IDE) IntelliJ Community Edition 2016.1.2 con el plugin de Scala.
* Tener instalado y configurado previamente una base de datos PostgreSQL. (sobre la máquina destinada a ser el servidor de datos)
* Tener instalado y configurado JMeter versión 3.0 (tanto en los esclavos como en el manejador general)

**Resultados de la entrega anterior:**

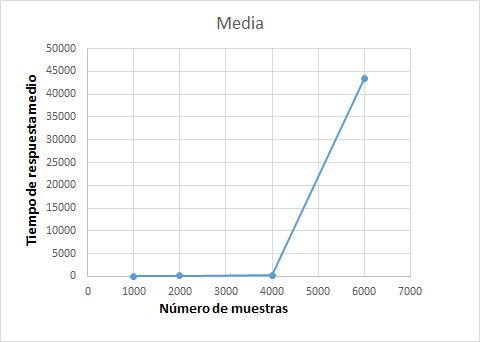
**Artefactos construidos**

* Cada una de las clases se construyó efectivamente en la aplicación, los cambios respecto a la planeación se dieron con respecto a la decisión de implementar la persistencia no mediante una clase “Mock” para cada entidad, sino mediante una base de datos configurada en postgresql 9.4 en una de las máquinas virtuales.
* Cada aplicación se conecta a la base de datos como el mismo usuario para realizar las consultas y la persistencia de los datos, cabe resaltar que este es un proceso asincrónico haciendo uso de los FutureTask.

**Análisis**



La gráfica que se muestra expone el número de muestras y la latencia que tuvo, donde se ve un crecimiento lineal en el comienzo y finaliza de forma tal que a partir de cierta cantidad de datos esta latencia puede llegar a ser constante. Lo que se hizo fue analizar el resultado de el número de registros que hace un sensor en un segundo. Lo que se busca es que el sistema responda a 4800 peticiones en un segundo, sin embargo esto no se logró ya que el tiempo fue mayor y en las últimas pruebas se encontró porcentajes de error. Los resultados obtenidos pueden verse afectados a causa de la latencia que hay en el envío de peticiones a la base de datos y el tiempo de respuesta que esta se demora.

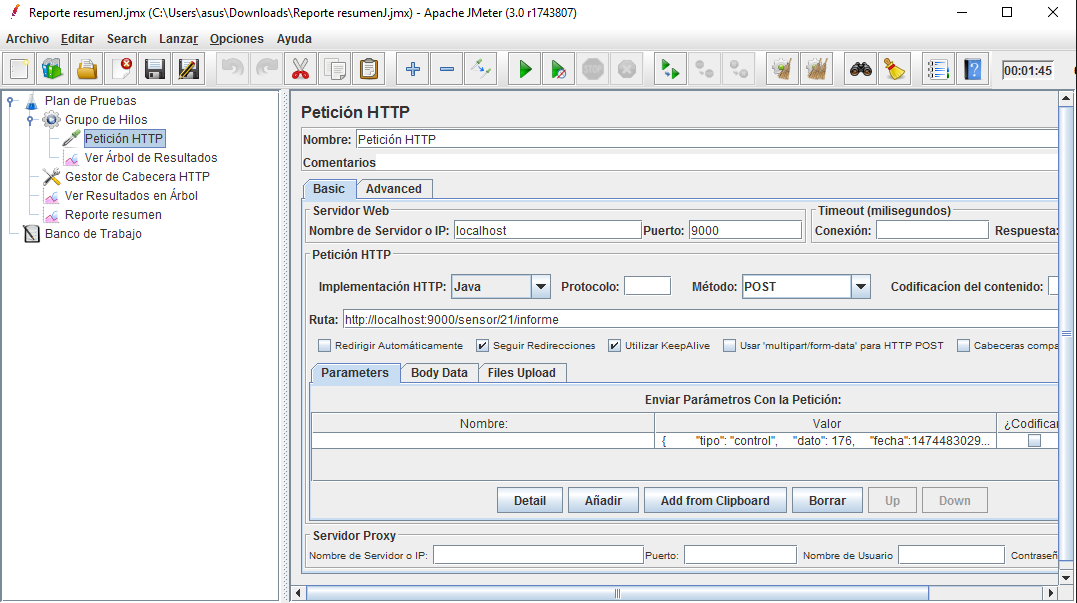


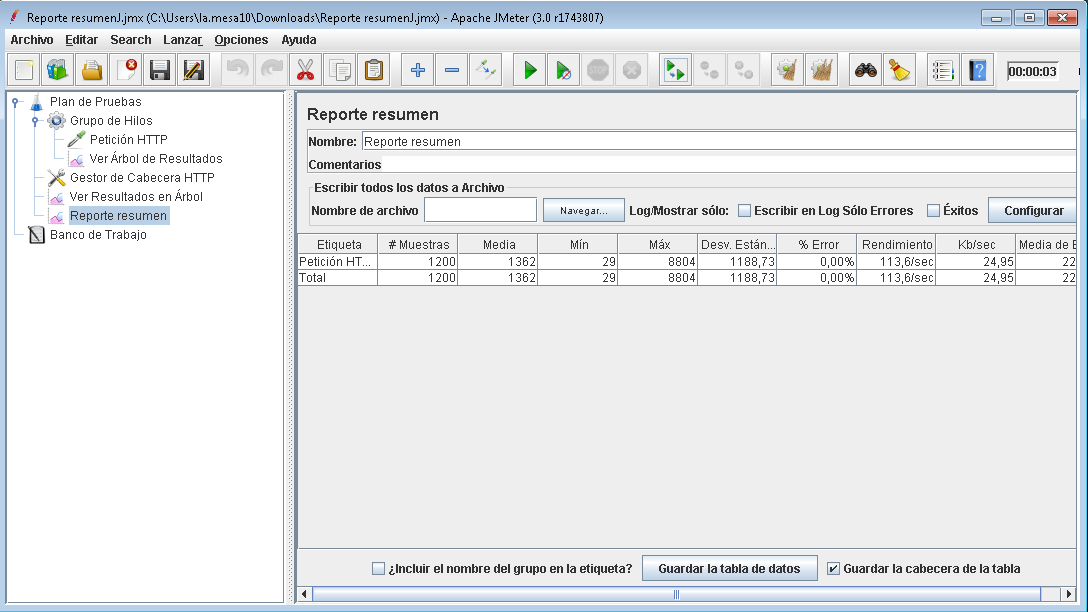
La gráfica anterior representa pruebas realizadas para estudiar el funcionamiento del método post de la entidad usuario respecto a el tiempo medio de respuesta y el número de muestras, estas se realizaron en una ventana de tiempo de 60 segundos, se puede observar el excelente comportamiento del software hasta antes de los 4000 hilos ejecutados en JMeter debido a su a que su tiempo de respuesta máximo fue de 282 milisegundos y así mismo los errores solo se presentan al sobrepasar las 4000 solicitudes.

**Resultados esperados**

La aplicación debe cumplir con los escenarios mínimos de calidad enunciados en las pruebas, al ejecutar todas las peticiones del CRUD de cada entidad eficazmente.

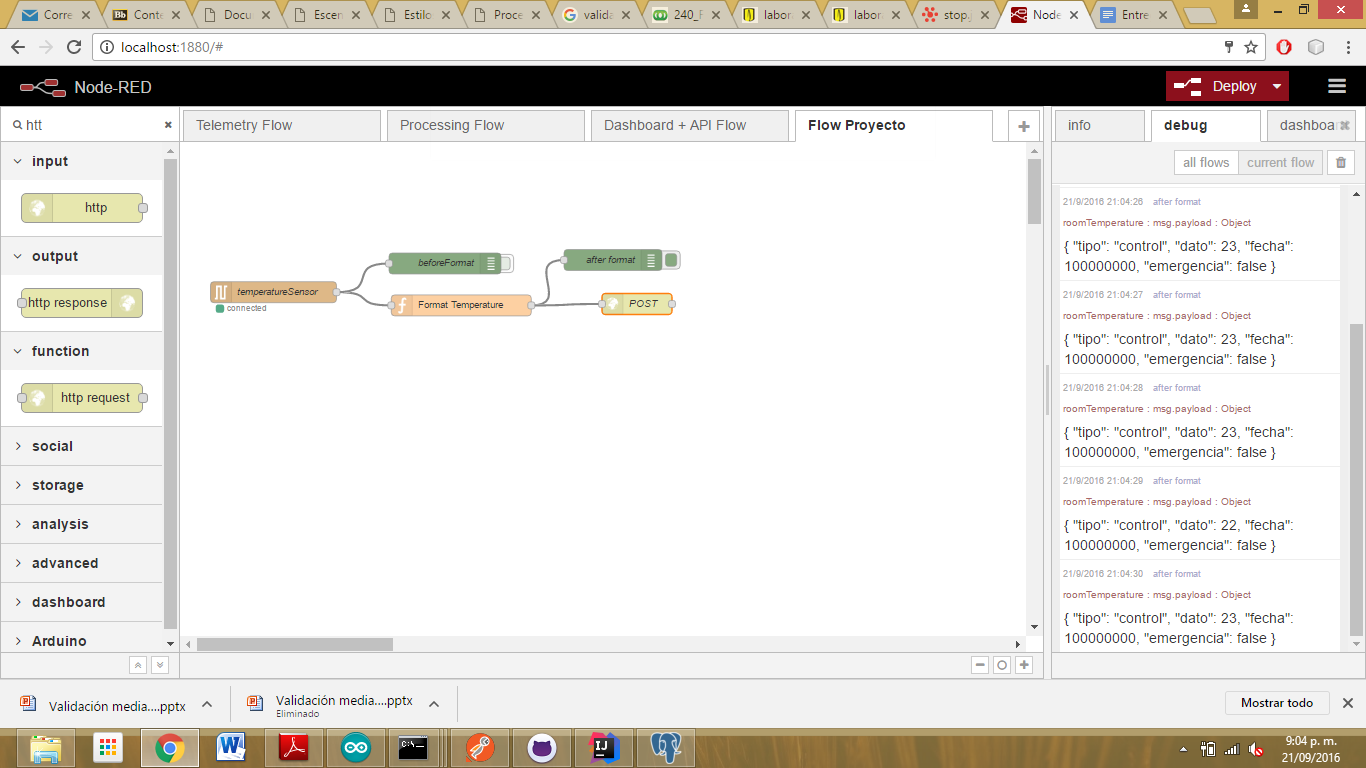
A continuación se detallan pruebas ejecutadas sobre la aplicación en una ventana de tiempo de un segundo, en esta se envían 1200 peticiones con el fin de crear y almacenar informes por partes de los sensores. Idealmente se espera que el software esté en capacidad de recibir toda la información de los sensores en un segundo con un porcentaje de error de 0% con la menor cantidad de tiempo medio posible. Como se puede observar en las imágenes las pruebas son ejecutadas siguiendo los parámetros apropiados y sus resultados demuestran que el software está preparado para recibir la información de 1200 informes por parte de un sensor, además se encuentra que el tiempo de respuesta medio es de 1360 ms.

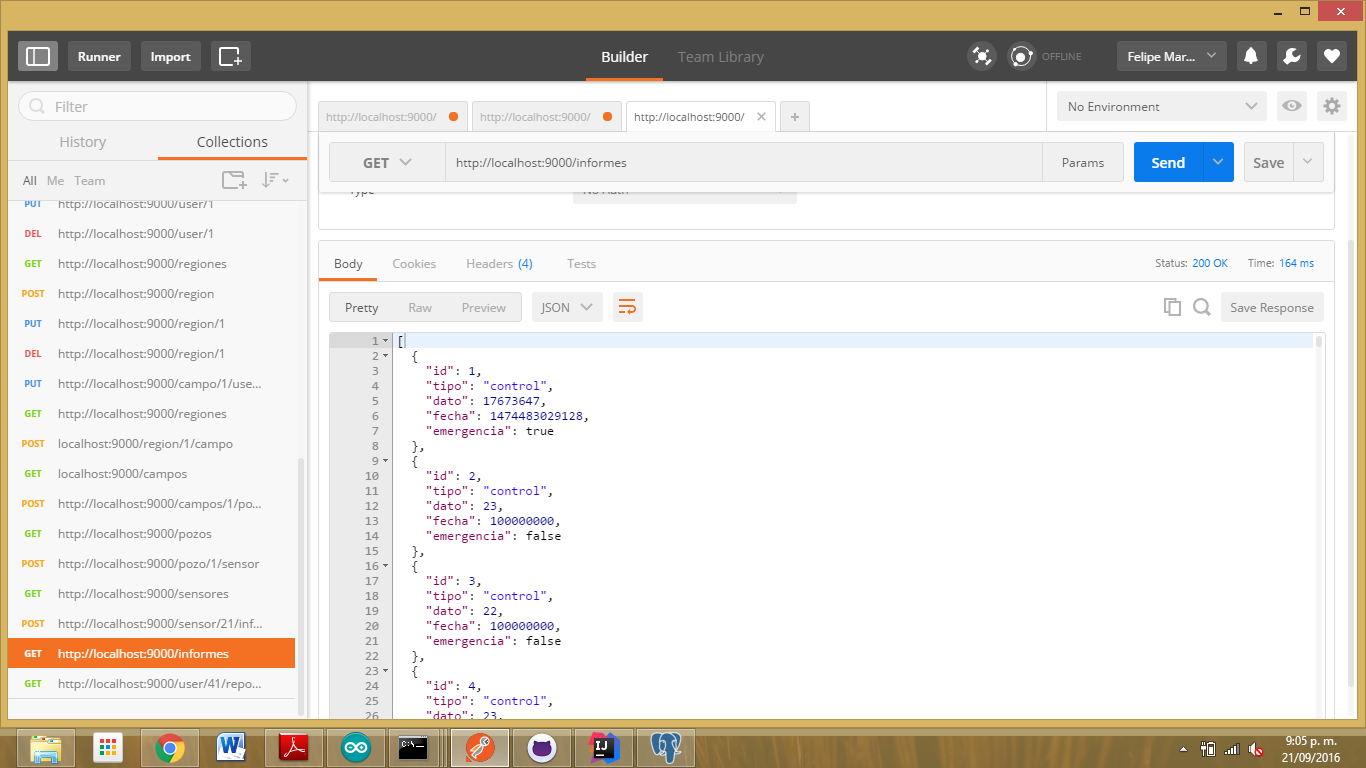




3. Envío de datos desde microcontrolador.

El sensor va generando JSON los cuales son consumidos por medio de un servicio rest. En este caso, el microcontrolador va haciendo post con la información generada la cual se va guardando en la base de datos. A continuación se podrá ver el esquema hecho en node-RED y una imagen de postman donde se hace un get de los informes generados por el sensor en el que se evidencia que los datos han sido guardados correctamente en la base de datos.





4. La lógica de negocio ofrecida por el sistema debe ser consumida por el microcontrolador.

Como se explicó anteriormente, el microcontrolador genera JSONs a partir de la información que le llega del sensor. A partir de esto, este utiliza los servicios de la aplicación y comienza a hacer POST de informes generado por un sensor. De esta forma la información se guarda en la base de datos para luego ser usado por los usuarios de la aplicación los cuales pueden pedir reportes de la información que necesitan. En este caso sería el jefe de campo y jefe de producción. Con lo anterior se entiende la lógica de negocio en la que entra el microcontrolador.

**Conclusiones:**

Al comparar los datos de la entrega anterior con esta, se logra mejorar los tiempo en la pruebas y disminuir el porcentaje de error a cero por-ciento, de esta forma se cumple con los requerimientos de escalabilidad y desempeño. Por lo anterior, se ve que los objetivos planteados de la entrega anterior sobre mejorar el desempeño de la arquitectura se lograron con éxito gracias a ciertos ajustes en la configuración del software.

**Repositorio de la aplicación**

[*https://github.com/f-martinez11/Oilcol*](https://github.com/f-martinez11/Oilcol)