**Documento de experimentación**

**Pre-experimentación**

Problemática

La cantidad de pacientes, médicos y trabajadores del hospital no afecta el tiempo de respuesta frente a emergencias, toma de información cardiaca, sincronización con el dispositivo móvil, reconfiguración del marcapasos, envío de reportes al hospital y de consejos.

Objetivo

Probar que la cantidad de usuarios que reciben o envían información no afecta al tiempo de respuesta.

Descripción

Para llevar a cabo este experimento se diseñará el modelo conceptual del sistema, con base en la descripción del experimento, para luego construir la lógica de negocio, la cual contiene las funcionalidades que deben ser ofrecidas en el servidor del hospital. A continuación, se desarrollarán las pruebas de carga con las cuales se procede a verificar la escalabilidad y desempeño del sistema.

Artefactos a construir

Se construirá la lógica del sistema por medio de controladores y modelos haciendo uso de la arquitectura Play.

Recursos de la experimentación

Para llevar a cabo este experimento se usarán máquinas virtuales con sistema operativo de 64 bits. También se usará IntelliJ Idea Ultimate en su versión 2016.3.4

Para integrar el trabajo se usará un repositorio en Github y para probar los servicios se hará uso de Postman.

Además, se hará uso de la arquitectura Play, con el fin de manejar la concurrencia de eventos por medio de dispatchers.

Resultados esperados

Se espera que el sistema sea capaz de recibir información de frecuencia cardiaca, presión sanguínea y nivel de estrés, también debe poder recibir notificaciones de emergencias, consultar historial de un paciente, enviar consejos y reconfiguración de marcapasos, manejar la información de los pacientes junto con su historia clínica y debe soportar la recepción de información desde 3000 sensores en un rango de tiempo de 1 segundo.

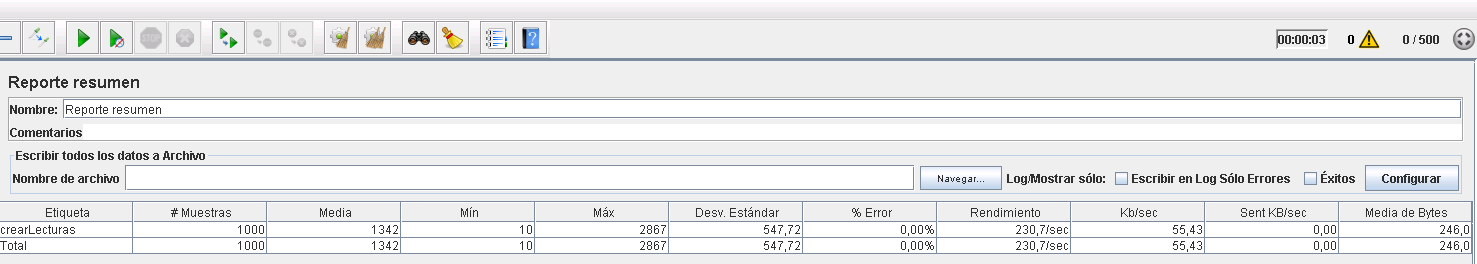
Duración y etapas

Se definen cuatro etapas. La primera es la etapa de análisis, en la cual se estudiará el enunciado del experimento y se definirán los requerimientos funcionales y no funcionales; esta etapa tomará una hora. La segunda etapa es la de diseño, en la cual se definirá el modelo conceptual y la arquitectura a utilizar; esta etapa durará tres horas. La tercera etapa es la de construcción de la lógica de negocio y tomará tres días. Por último, está la etapa de pruebas en la que se probará que el sistema soporte la recepción de información; esta etapa tomará tres horas.

**Experimentación**

**Post-experimentación**

Resultados obtenidos



Los resultados obtenidos se pueden evidenciar a través de la tabla anterior, en la cual se puede observar que la media de respuesta es de 1342 milisegundos, con un rendimiento de 230,7 threads ejecutados por segundo con un 0% de error, lo que indica que todas las peticiones fueron atendidas exitosamente.



Adicionalmente, se puede ver mejor en la gráfica que el rendimiento fue mejorando cuando la cantidad de threads aumentaba hasta llegar a mil peticiones, además la desviación es baja lo que indica que la distancia entre los diferentes tiempos de respuesta no fue tan grande.

Duración real

El tiempo teórico de análisis del enunciado del proyecto fue una hora y el tiempo experimental fue también de una hora. Por otro lado, en la segunda etapa el modelo conceptual y la arquitectura a utilizar se demoró 2:30 horas cuando el tiempo teórico fue de 3 horas. Además, la tercera etapa es la de construcción de la lógica de negocio que tomo 4 días cuando teóricamente se decidió que tomaría tres días. Finalmente, la cuarta etapa de pruebas nos llevó aproximadamente un día, cuando habíamos estipulado 3 horas.

Artefactos construidos

En nuestro proyecto implementamos el estilo de arquitectura por actores a través de Play, por esto manejamos dos paquetes principales, uno de controladores, y el otro de modelos (Entities). Estos paquetes se construyeron para manejar la lógica del negocio de manera que se acoplara bien con la tecnología de Play, además se hizo una conexión con la base de datos Postgres para manejar la persistencia y poder probar el funcionamiento del proyecto. No se implementó ninguna otra arquitectura ni artefacto, dado que solamente con estos paquetes se seguía el lineamiento de Play, manejo de concurrencia por medio de dispatchers.

Análisis

Uno de los motivos por los que el tiempo medio no fue mejor se puede deber a la conexión a la base de datos, dado a que toma mucho tiempo acceder a la memoria secundaria a recuperar los registros, y Play ya tiene la información en la RAM. Por otro lado, se usaron dos tácticas, una es la introducción a la concurrencia ya que se procesa en paralelo los eventos con threads, y el otro, mantener múltiples copias de computaciones, es decir, la introducción de dispatcher que controlan la división de la carga. De igual forma usamos FIFO en la táctica de Schedulling Policy para así, tratar las solicitudes como iguales, sin problema de que alguna tenga más prioridad que otra. Uno de los principales problemas es que en la prueba de carga se logra crear todos los threads en un segundo pero toma más tiempo responder a las solicitudes ya que son solicitudes de POST, por lo tanto, la base de datos tiene que hacer revisión de restricciones, candados, entre otros.

Conclusiones

En conclusión la hipótesis planteada fue puesta a prueba y se obtuvieron resultados positivos, es decir, se comprobó que la cantidad de usuarios y solicitudes que recibía el sistema no tuvieron una alta influencia en la latencia (tiempo de respuesta entre el momento que se recibe la solicitud y en que se responde). Algunas recomendaciones son disminuir el acceso a la base de datos lo más que sea posible, también, es importante diseñar el modelo relacional muy bien desde el principio, para evitar que el nivel de acoplamiento sea alto.