**EXPERIMENTO 1: ESCENARIOS DE CALIDAD DE ESCALABILIDAD Y DESEMPEÑO**

**Pre-experimentación**

1. **Problemática: Se desea construir un sistema reactivo de información cardiaca para el Hospital Cardiológico de Santa Fe. Este sistema se pondrá a prueba en su capacidad para tener un crecimiento continuo de trabajo sin perder su reactividad ni su calidad en los servicios que ofrece (escalabilidad). Igualmente, se medirá su capacidad de respuesta ante peticiones sin perder calidad en sus servicios (desempeño).**
2. **Objetivo: Analizar las decisiones de diseño del sistema para cumplir con los escenarios de calidad especificados: escalabilidad y diseño.**
3. **Descripción del experimento:**

**Según el enunciado entregado, se planean realizar las siguientes actividades:**

* **Definir casos de uso de la aplicación y actores**
* **Realizar un modelo del problema (UML)**
* **Definir estilos arquitecturales a implementar en la prueba del experimento**
* **Documentar la pre-experimentación**
* **Implementar el sistema desarrollado**
* **Realizar pruebas unitarias**
* **Realizar pruebas de carga en JMeter**
* **Analizar resultados con base a los escenarios de calidad**
* **Aceptar o rechazar la hipótesis planteada**
* **Balance de los resultados y oportunidades de mejora**
* **Conclusiones del experimento**

**Con base en estas actividades, se han de obtener los siguientes datos:**

* **Medidas de latencia media en respuestas a peticiones por n muestras**
* **%Error frente a peticiones de servicios**
* **Rendimiento (n solicitudes atendidas por segundo)**
* **Número de muestras de soporta el sistema**

1. **Artefactos a construir:**

**Para este experimento se construirá una aplicación web con stateless beans, la cual será síncrona y expondrá sus servicios con una capa REST.**

1. **Recursos de la experimentación:**

* **Ambiente de desarrollo Netbeans IDE 8.2**
* **API de Java JAX-RS para servicios REST**
* **Apache JMeter Versión 3.1**

1. **Resultados esperados:**

**Con el fin de determinar la escalabilidad y desempeño de la solución propuesta, se realizarán pruebas de carga JMeter sobre el requerimiento funcional de recepción de información de sensores para el caso de prueba, en donde se busca tener los siguientes resultados:**

* **Latencia media menor o igual a 1000 ms hasta 3000 muestras mínimo.**
* **% Error de 0% para 3000 muestras mínimo.**
* **Ramp-up de 60 segundos con resultados anteriores de forma satisfactoria**

1. **Duración y etapas:**

|  |  |
| --- | --- |
| **Etapa** | **Tiempo** |
| **Revisión de requerimientos funcionales y selección de estilo arquitectural: JAX-RS** | **1 hora 30 minutos** |
| **Implementación capa lógica** | **3 horas** |
| **Implementación capa servicios** | **1 hora** |
| **Pruebas de Carga** | **1 hora** |
| **Análisis pruebas de Carga** | **30 minutos** |
| **Conclusiones Experimento** | **10 minutos** |

**Post-experimentación**

1. **Resultados obtenidos:** Se realizaron pruebas de carga para la recepción de información de sensores con la siguiente decisión de diseño: cada sensor recibe información en un servicio independiente, de esta manera no se sobresaturan las lecturas de sensor y es posible solicitar lecturas específicas del estado del paciente.

En total son 3 servicios de creación de información para cumplir la funcionalidad de recepción de información de sensores, los cuales se representan como peticiones HTTP de POST:

* crearMedidaSensorPresion: Refiere a las medidas de presión sanguínea del paciente, cuyas peticiones por #muestras (samples), latencia media (avg), latencia mínima, latencia máxima y % error se registraron en la ejecución de JMeter:



* crearMedidaSensorFreq: Refiere a las medidas de frecuencia cardiaca del paciente, cuyas peticiones por #muestras (samples), latencia media (avg), latencia mínima, latencia máxima y % error se registraron en la ejecución de JMeter:
* crearMedidaSensorEstrés: Refiere a las medidas de estrés del paciente, cuyas peticiones por #muestras (samples), latencia media (avg), latencia mínima, latencia máxima y % error se registraron en la ejecución de JMeter:

**Resultados obtenidos con la capa de persistencia:**

Antes de implementar la capa de persistencia, se decidió cambiar el API de JAX-RS por el API de Play, con el fin de establecer una arquitectura asincrónica en donde se establezcan prioridades en la ejecución de tareas (Manejo de ThreadPools y solicitudes según el modelo de actores).

* crearMedidaSensorPresion: Refiere a las medidas de presión sanguínea del paciente, cuyas peticiones por #muestras (samples), latencia media (avg), latencia mínima, latencia máxima y % error se registraron en la ejecución de JMeter:



* crearMedidaSensorFreq: Refiere a las medidas de frecuencia cardiaca del paciente, cuyas peticiones por #muestras (samples), latencia media (avg), latencia mínima, latencia máxima y % error se registraron en la ejecución de JMeter:



* crearMedidaSensorEstrés: Refiere a las medidas de estrés del paciente, cuyas peticiones por #muestras (samples), latencia media (avg), latencia mínima, latencia máxima y % error se registraron en la ejecución de JMeter:



1. **Duración real:**

|  |  |  |
| --- | --- | --- |
| **Etapa** | **Tiempo Teórico** | **Tiempo real** |
| **Revisión de requerimientos funcionales y selección de estilo arquitectural: JAX-RS** | **1 hora 30 minutos** | **1 hora 30 minutos** |
| **Implementación capa lógica** | **3 horas** | **4 horas** |
| **Implementación capa servicios** | **1 hora** | **2 horas** |
| **Pruebas de Carga** | **1 hora** | **40 minutos** |
| **Análisis pruebas de Carga** | **30 minutos** | **30 minutos** |
| **Conclusiones Experimento** | **10 minutos** | **15 minutos** |
| **Selección e investigación de API Play para nueva arquitectura** | **3 horas** | **4 horas** |
| **Implementación lógica y servicios Play** | **5 horas** | **5 horas** |
| **Implementación persistencia** | **2 horas** | **3 horas** |
| **Pruebas de Carga** | **1 hora** | **40 minutos** |
| **Análisis pruebas de Carga** | **30 minutos** | **30 minutos** |
| **Conclusiones Experimento** | **10 minutos** | **15 minutos** |

1. **Artefactos construidos:** Capa de lógica (backend) y uso de Mocks, Capa de servicios, en arquitectura JAX-RS. Pruebas de carga en JMeter. Capa de persistencia, lógica y servicios en Play.
2. **Análisis:** Con basea las tablas de datos proporcionados por JMeter en cada prueba, se puede apreciar que el sistema soportó hasta 9000 muestras sin reportar ningún error y con una latencia menor a 1 segundo (menor a 1000 ms en mínimo, promedio y máximo). Esto quiere decir que el sistema tiene una escalabilidad muy razonable para soportar 9000 pacientes potenciales a los cuales constantemente se les debe registrar su estado de salud. Adicionalmente, el desempeño es admirable en esta aplicación debido a que la configuración de Ramp-up determinada para todas las pruebas es de 60 segundos; tiempo en el cual se lanzan e inician los threads para un determinado número de muestras. En el caso de mayor muestra:

**(Ramp up period) / (Number of threads) = 60s/9000 Threads = 0,0066 segundos.**

Lo anterior quiere decir que se enviaron peticiones de sensores al sistema cada 0,0066 segundos sin reportar ningún error, lo cual es un índice de respuesta rápido y muy importante para el cuidado de pacientes en la solución del hospital, ya que el sistema atiende peticiones en periodos muy cortos de tiempo y no se satura con las lecturas de los sensores.

A continuación, se presentan los datos de latencia (promedio, mínima y máxima) por número de threads en la ejecución de los servicios de creación de medidas de cada uno de los sensores, los cuales en conjunto soportan hasta 9000 pacientes potenciales sin reportar errores:

**Servicio POST que registra la lectura de un sensor para determinar el estrés de un paciente con respecto a una escala específica. Se pueden apreciar picos incrementales en la latencia máxima, posiblemente debidas a la capacidad de JMeter al crear y lanzar los threads, en donde cada vez tarda un poco más debido a que el servidor comienza a saturarse.**

**Servicio POST que registra la lectura de un sensor para determinar la frecuencia cardiaca de un paciente con respecto a una escala específica. Se pueden apreciar picos variantes en las muestras iniciales, posiblemente debido a la capacidad de JMeter al crear los threads en donde se genera un “pico de botella” en el muestreo inicial, que tiende a estabilizarse en el muestreo final.**

**Servicio POST que registra la lectura de un sensor para determinar la presión sanguínea de un paciente con respecto a una escala específica. Se pueden apreciar picos variantes en las muestras iniciales, posiblemente debido a la capacidad de JMeter al crear los threads en donde se genera un “pico de botella” en el muestreo inicial, que tiende a estabilizarse en el muestreo final.**

**Resultados Arquitectura Play**

**Servicio POST que registra la lectura de un sensor para determinar la frecuencia cardiaca de un paciente con respecto a una escala específica. Se pueden apreciar picos variantes en las muestras iniciales, posiblemente debido a la capacidad de JMeter al crear los threads en donde se genera un “pico de botella” en el muestreo inicial, que tiende a estabilizarse en el muestreo final.**

**Servicio POST que registra la lectura de un sensor para determinar el estrés de un paciente con respecto a una escala específica. Se pueden apreciar picos incrementales en la latencia máxima, posiblemente debidas a la capacidad de JMeter al crear y lanzar los threads, en donde cada vez tarda un poco más debido a que el servidor comienza a saturarse.**

**Servicio POST que registra la lectura de un sensor para determinar la presión sanguínea de un paciente con respecto a una escala específica. Se pueden apreciar picos variantes en las muestras iniciales, posiblemente debido a la capacidad de JMeter al crear los threads en donde se genera un “pico de botella” en el muestreo inicial, que tiende a estabilizarse en el muestreo final.**

1. **Conclusiones:** De este experimento se puedeconcluir que las decisiones de diseño empleadas en la solución para cumplir los atributos de diseño fueron satisfactorias, debido a que el sistema logró soportar hasta 9000 pacientes sin presentar errores, con entradas de peticiones cada 0,0066 segundos y con una latencia promedio de entre 1-5 milisegundos por cada sensor. Esto es debido a la implementación del estilo arquitectural por capas y el uso del API JAX-RS, los cuales presentaron las ventajas de conocimiento previo, escalabilidad y desempeño razonable para soportar al menos 3000 usuarios. Lo anterior se debe a la implementación de servicios simples e independientes que no requieren priorización ni muchas restricciones de negocio a comparación de arquitecturas orientadas a actores (Play, por ejemplo) en donde se encolan peticiones y se favorece la realización de tareas asíncronas.