1. **Enunciado Proyecto.**

El hospital Cardiológico de Santa Fe es una institución privada en Bogotá con más de 30 años de experiencia en el diagnóstico, cuidado y manejo de enfermedades cardiacas. El hospital es muy eficiente en el área de emergencias y cirugías, su equipo ha tratado cientos de casos complejos y es referente a nivel nacional en el área de cardiología.

**PROBLEMA**

Con toda su experiencia, las directivas desean mejorar su división de prevención, rehabilitación y cuidados externos. Actualmente, el hospital sólo dispone de 500 camas para pacientes en diversos grados de salud y atiende alrededor de 1000 pacientes al mes, los cuales pueden estar internados o seguir algún tratamiento posoperatorio o de cuidados ambulatorios desde sus hogares.

**DESCRIPCIÓN DEL SISTEMA A CONSTRUIR**

Los pacientes en postoperatorio o rehabilitación deben ir al hospital una vez por semana para realizarse un conjunto de exámenes de acuerdo a su cardiopatía y así saber el estado de salud actual y ajustar el tratamiento de ser necesario.

El hospital ve este proceso como algo dispendioso y complicado ya que cerca del 50 % de sus pacientes tiene que cambiar el horario de sus exámenes durante la semana y casi un 80 % de ellos cancela una cita por lo menos una vez al mes. La inasistencia afecta el plan de rehabilitación de los pacientes y los pone en riesgo de recaídas de salud o nuevas condiciones producto de la desatención.

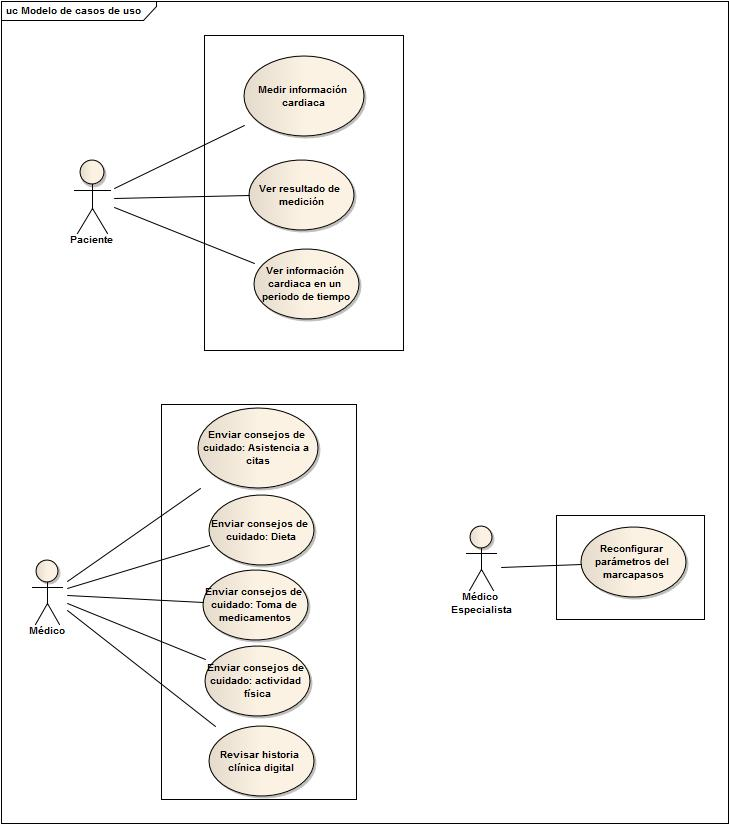
El hospital, se ha aliado con Thing Worx S.A.S. para darle a sus pacientes en rehabilitación unos brazaletes inteligentes que recopilan información cardiaca relevante para las diferentes cardiopatías (i.e., frecuencia cardiaca, presión sanguínea, nivel de estrés). Desde un punto de vista operacional los componentes del sistema interactúan como sigue:

* El dispositivo le recuerda al paciente que debe medir la información cardíaca
* Una vez hecha la medición, el brazalete se sincroniza con un dispositivo móvil del paciente con el fin de:
  + Mostrar al paciente dicha información y un dictamen que se encuentra en cualquiera de 3 niveles: rojo (requiere atención médica de urgencia ya que es un paciente de alto riesgo, i.e., recientemente infartado o con cirugía mayor), amarillo (debe darse un consejo relacionado a la higiene de vida, e.g., relajarse y alejarse de una situación de estrés cuando se detecte un ascenso en la tensión ) y verde (todo está bien, siga con su higiene de vida). De igual manera, debe ser posible ver la información cardíaca reportada en un período de tiempo a través de gráficas simples en el dispositivo.
  + Reportar al hospital la información cardíaca del paciente periódicamente.
  + Cuando la condición de salud se encuentra en nivel rojo, el dispositivo debe notificar a la ambulancia, los servicios de emergencia del hospital y al médico tratante de la ubicación (GPS) y condición actual del paciente para coordinar la atención y tomar el menor de tiempo posible para llegar a donde el paciente y atenderlo. La emergencia debe quedar guardada en el sistema.
* A partir de la información cardíaca reportada, los médicos pueden enviar a los dispositivos consejos de cuidado relacionados a dieta, actividad física, toma de medicamentos, asistencia a citas médicas. Sólo los médicos especialistas autorizados pueden reconfigurar los parámetros de los marcapasos para regular el ritmo cardíaco de un paciente, primero se reporta al dispositivo inteligente y luego de éste al marcapasos.
* Cada paciente puede tener uno o más de un médico asignado en cualquier momento de su tratamiento y que cada una de las decisiones, sea tratamientos, exámenes o diagnósticos deben estar completamente documentados dentro de la historia clínica digital, la cual se utiliza como sustento para el pago y notificación de procedimientos con las EPS.

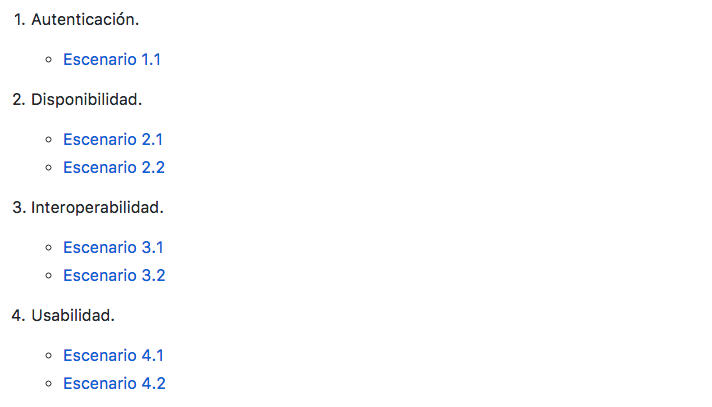
**2. Restricciones.**

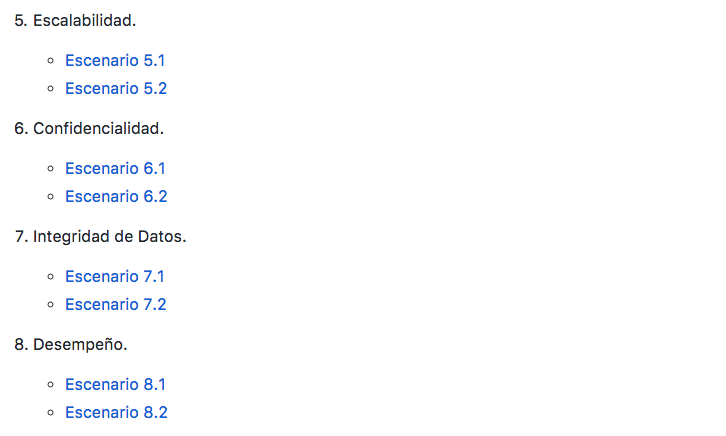
|  |  |  |
| --- | --- | --- |
| **Identificador** | **Tipo** | **Descripción** |
| **RE-01** | **Tecnología** | El servicio central debe estar desplegado en un servidor Netbeans. |
| **RE-02** | **Negocio** | Solo el Médico Especialista puede modificar los parámetros del marcapasos. |
| **RE-03** | **Tecnología** | El servicio móvil debe estar desplegado en un servidor de aplicaciones. |
| **RE-04** | **Tecnología** | El dispositivo móvil debe contar con tecnología bluetooth. |
| **RE-05** | **Negocio** | El proyecto debe ser entregado antes del 24/04/2017 |
| **RE-06** | **Tecnología** | El dispositivo móvil debe contar con GPS. |

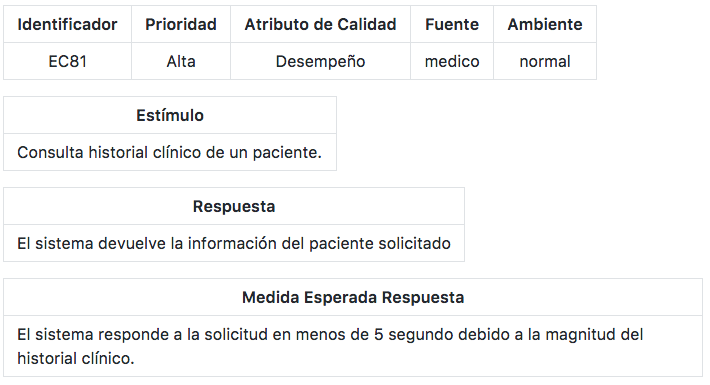
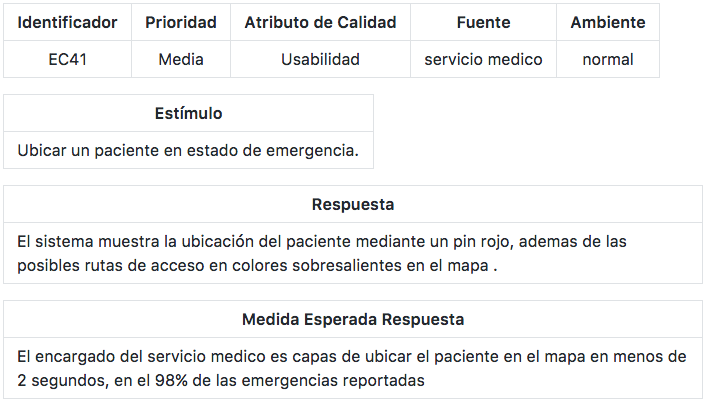
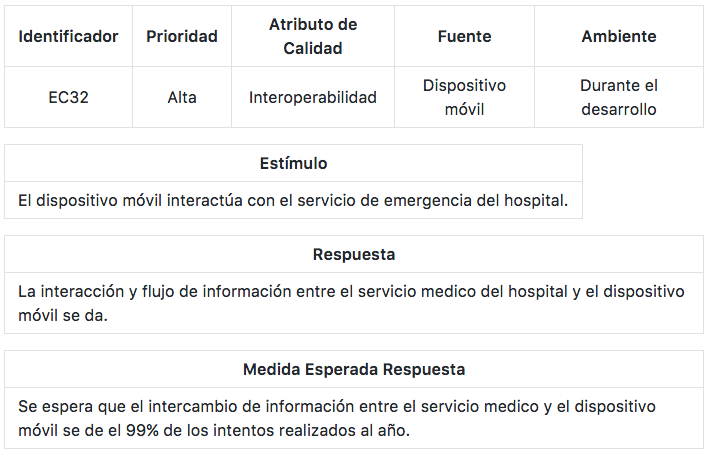
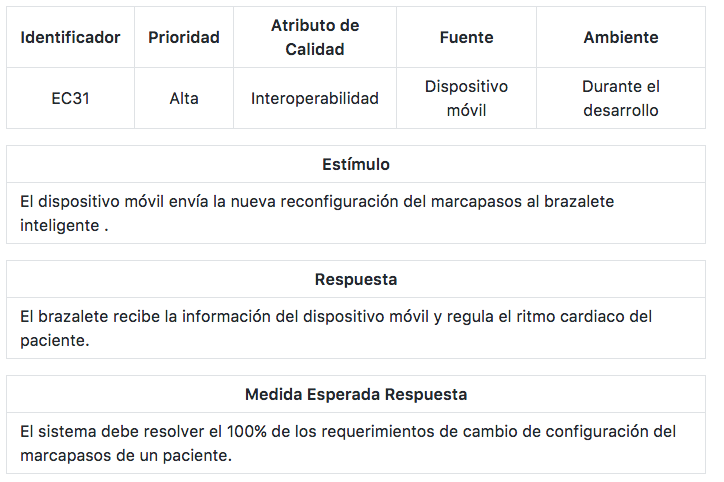
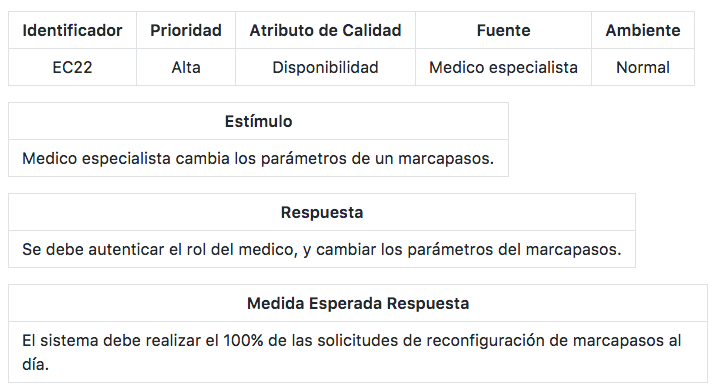
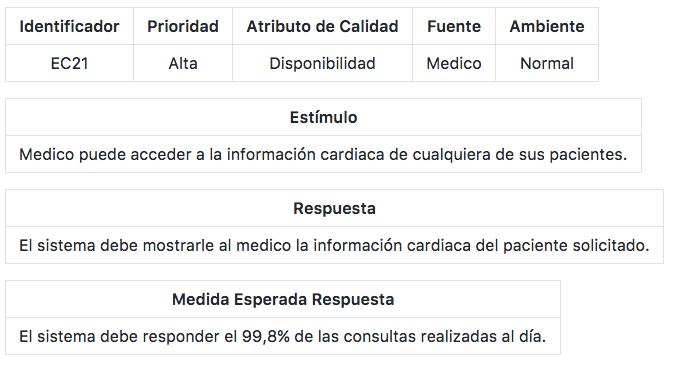
**3. Casos de uso**



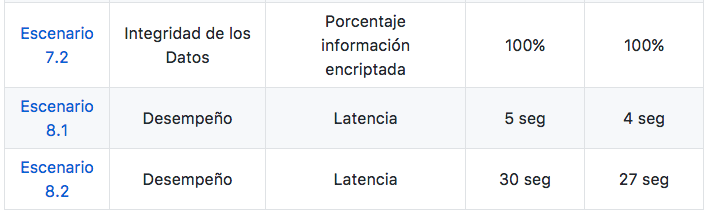
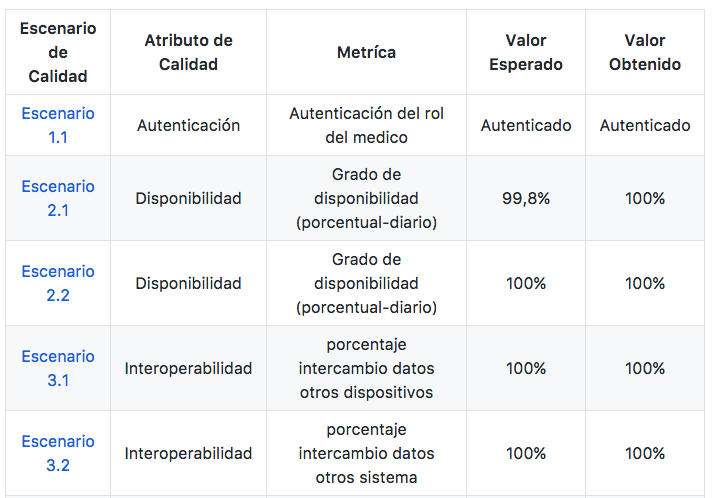
**4.Atributos de Calidad.**





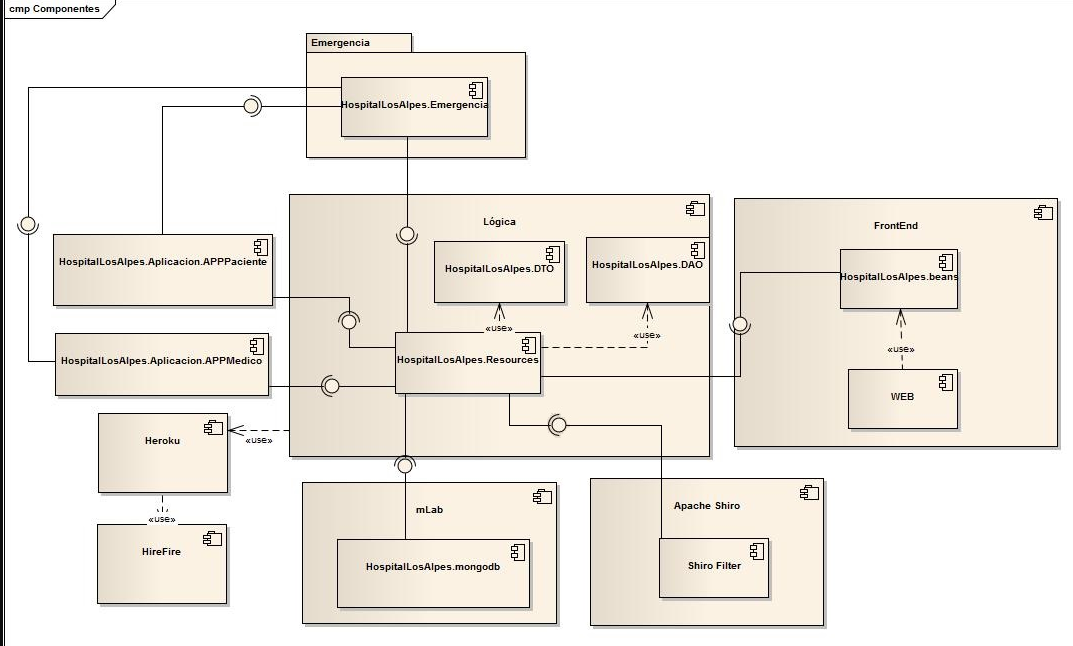


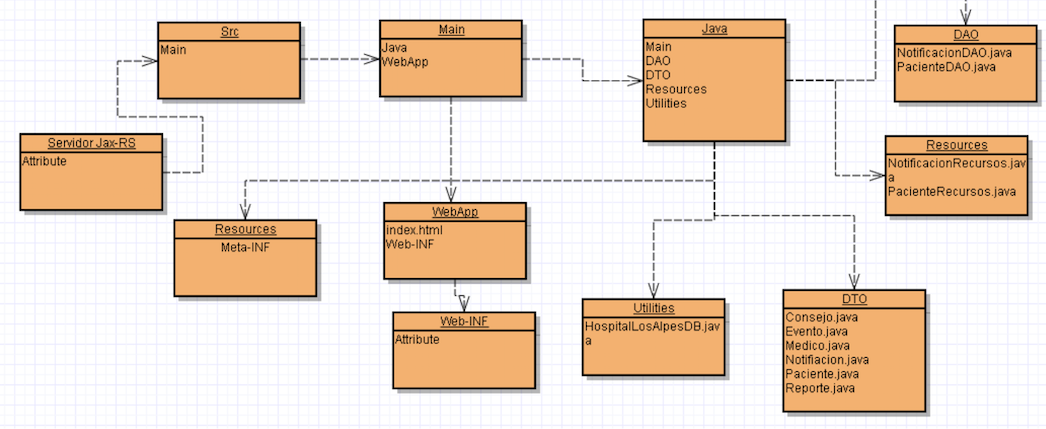
**4.Resumen métricas.**



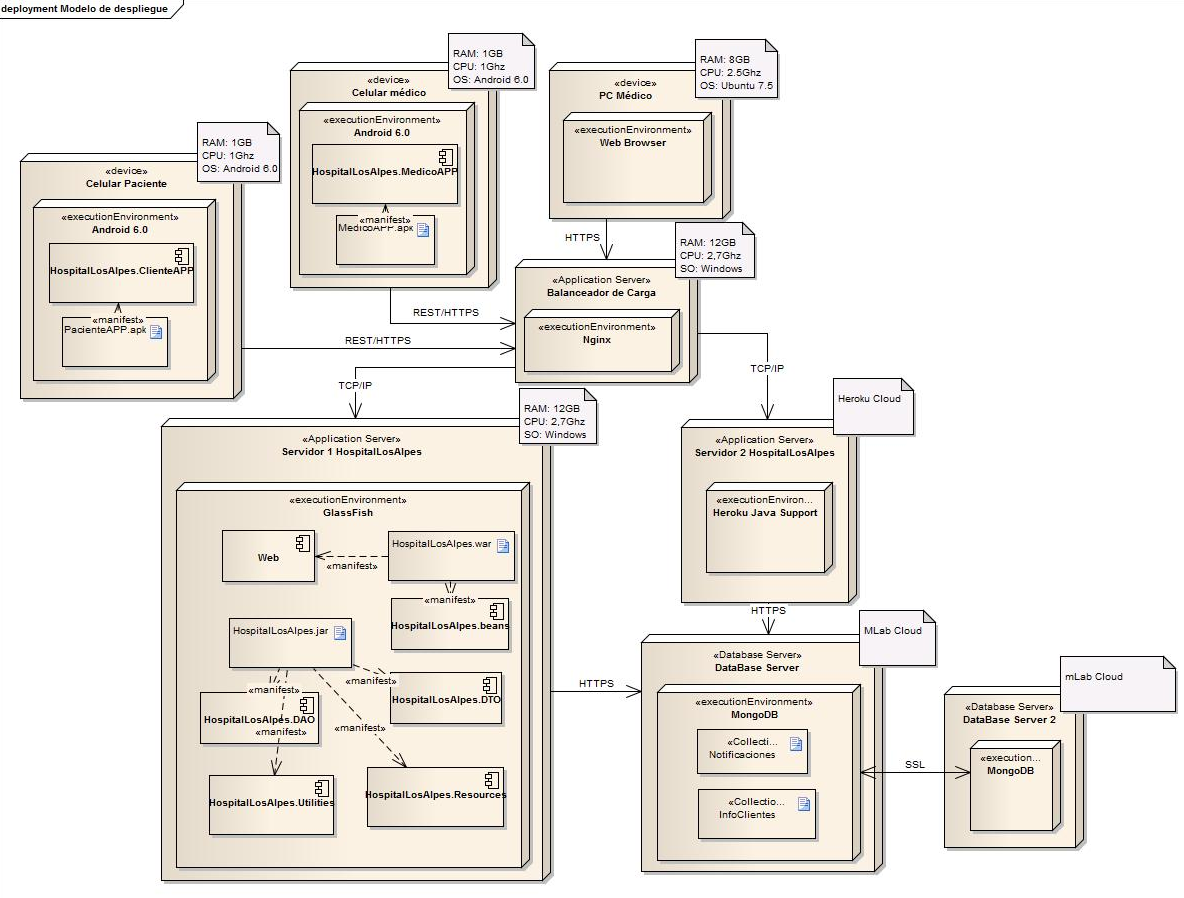
**5. Vistas Arquitecturales**

**Diagrama de componentes**

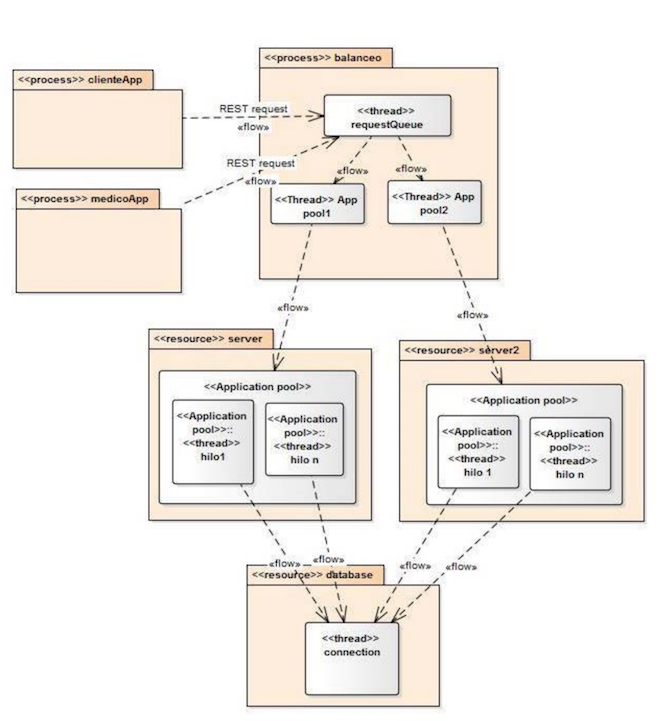


**Diagrama de desarrollo**

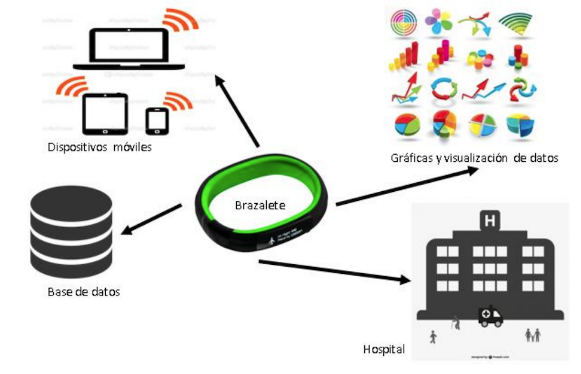
**Diagrama de despliegue**



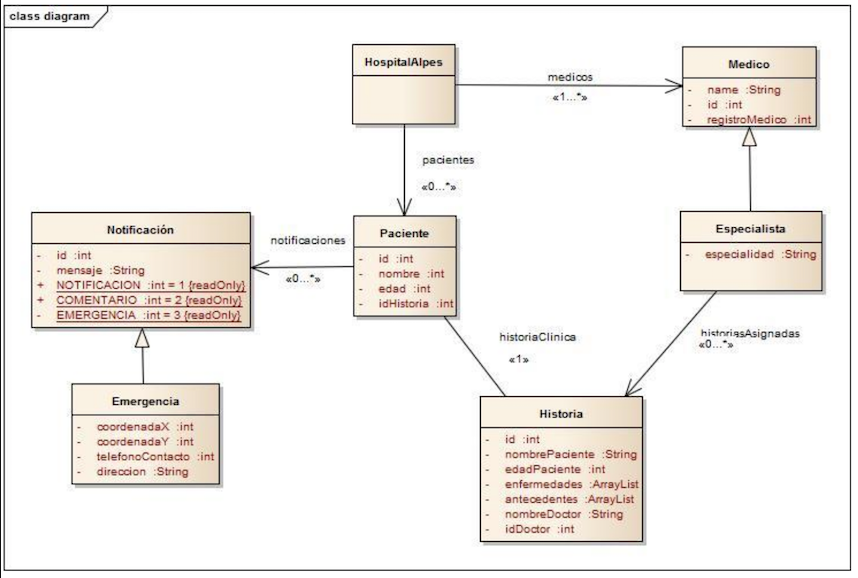
**Diagrama de concurrencia**



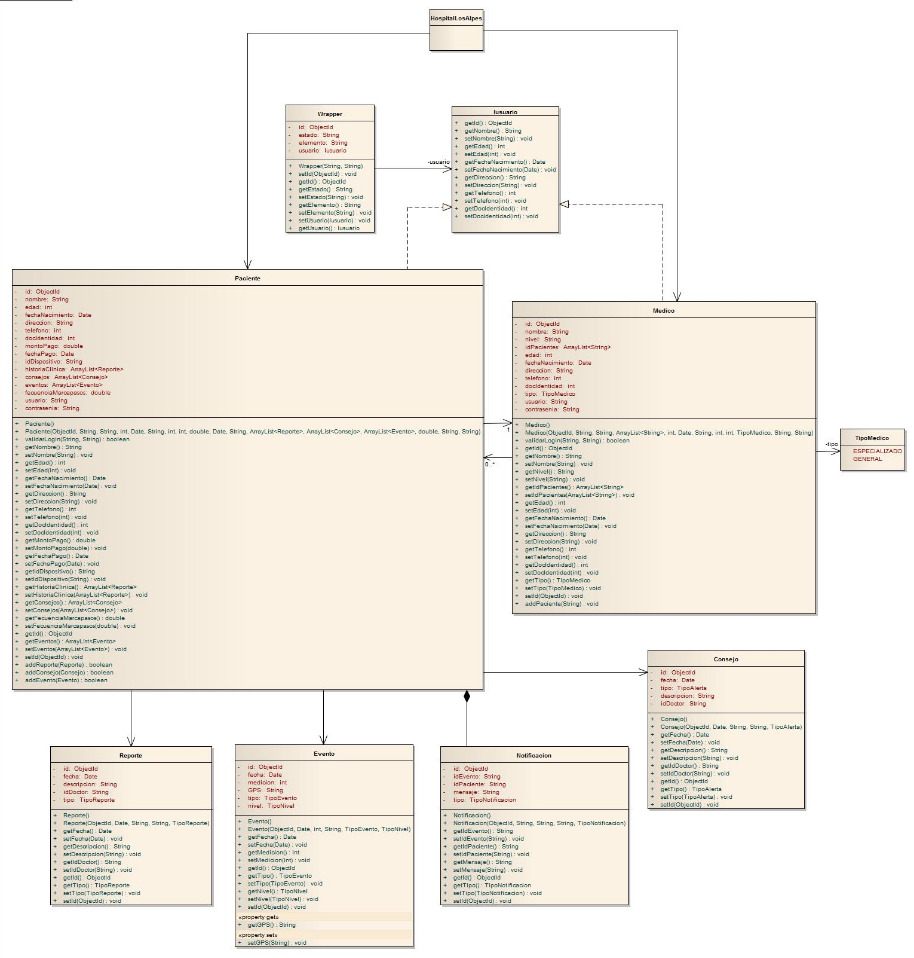
**Diagrama de contexto:**



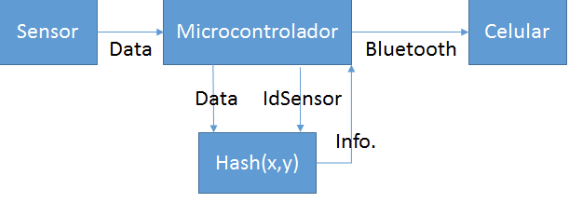
**Diagrama de información:**



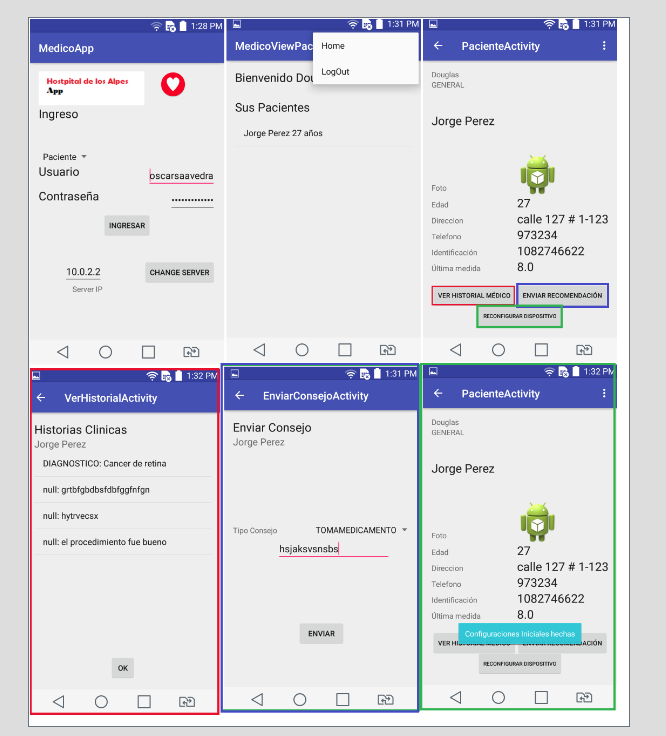
**Modelo de clases:**



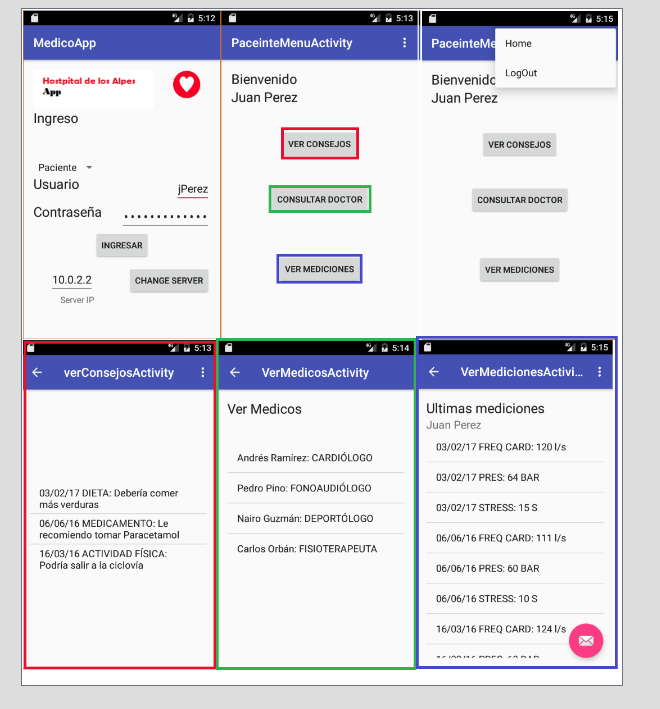
**Seguridad del sensor:**



**Navegabilidad del medico:**

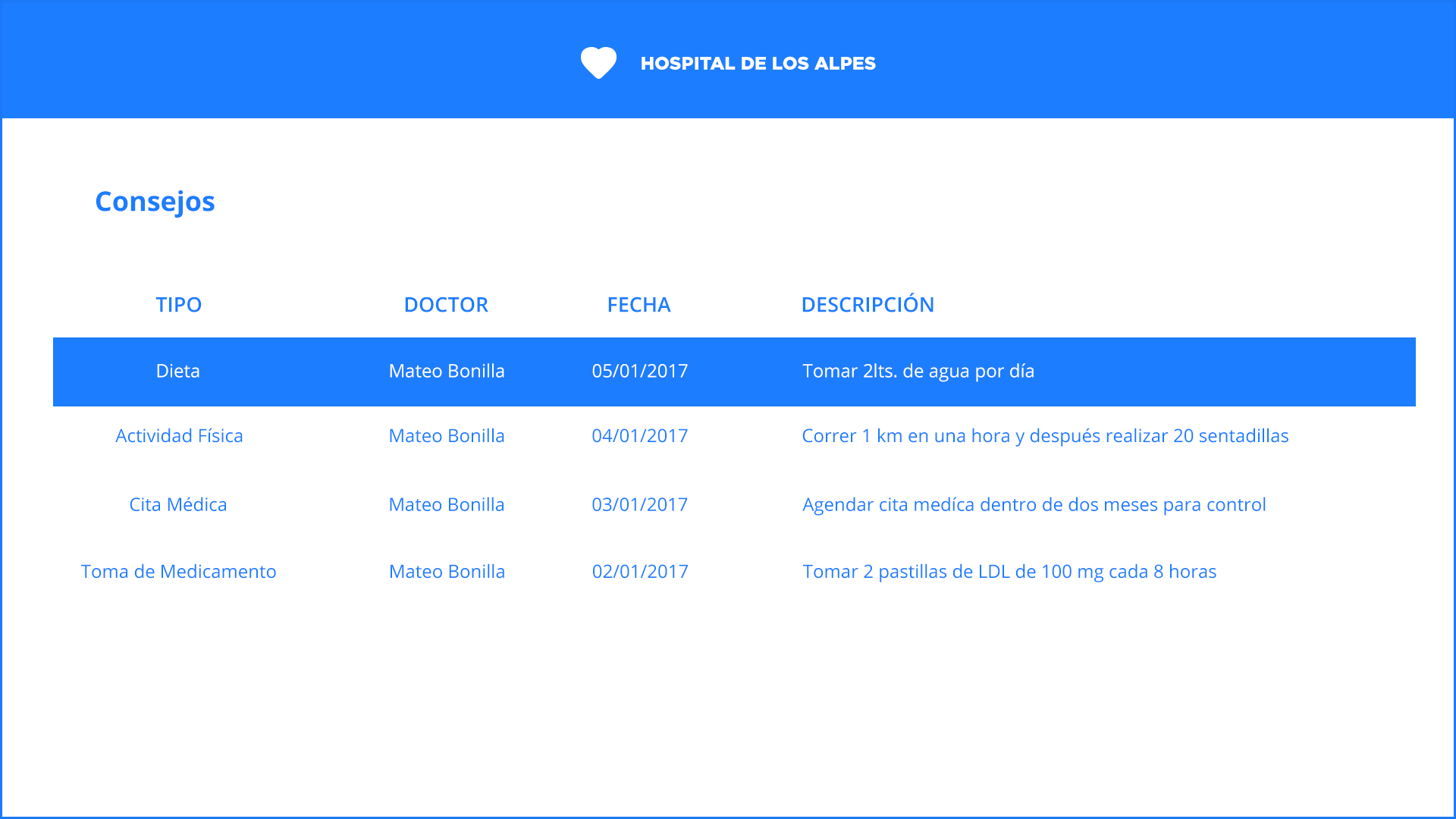
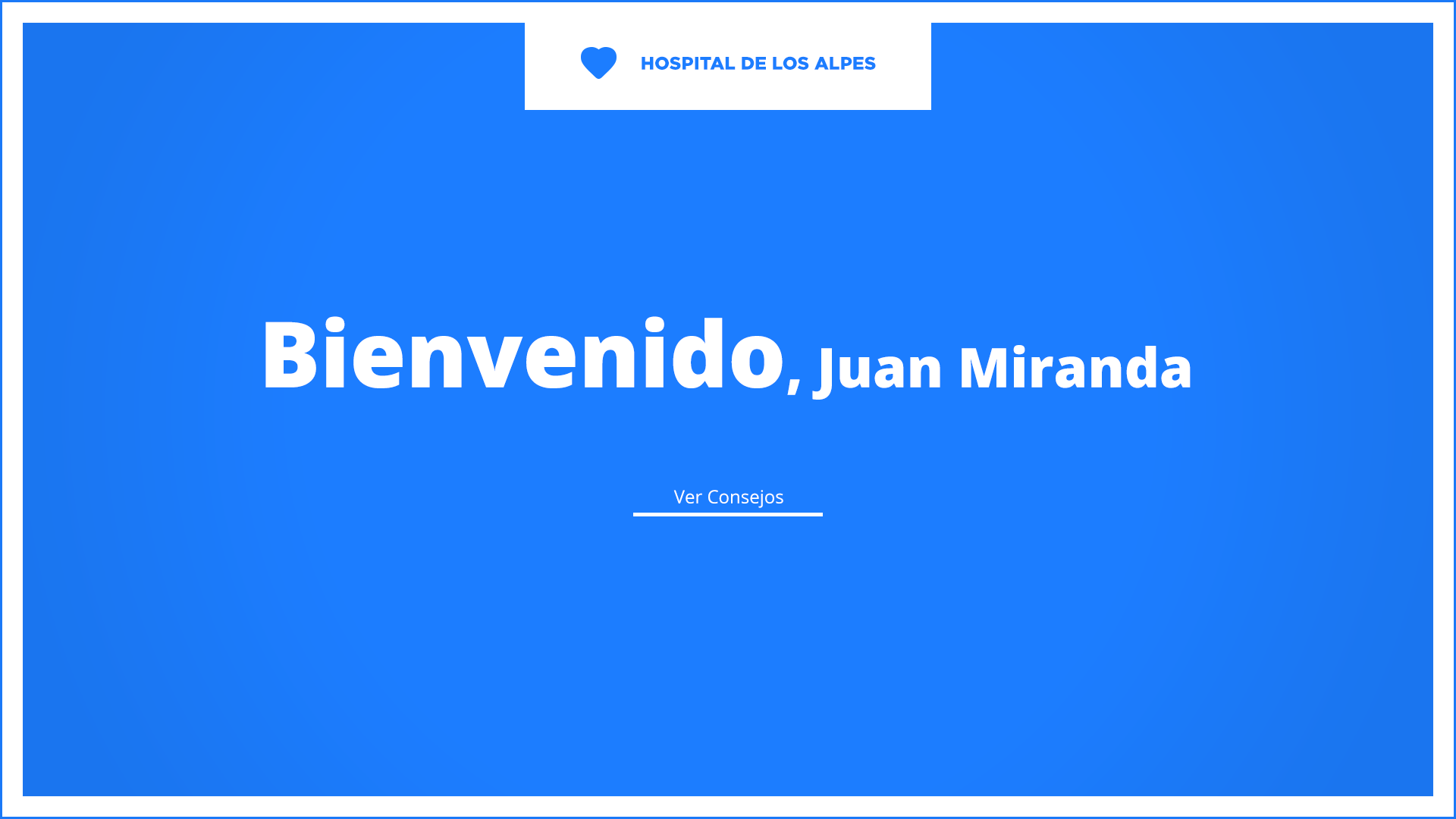
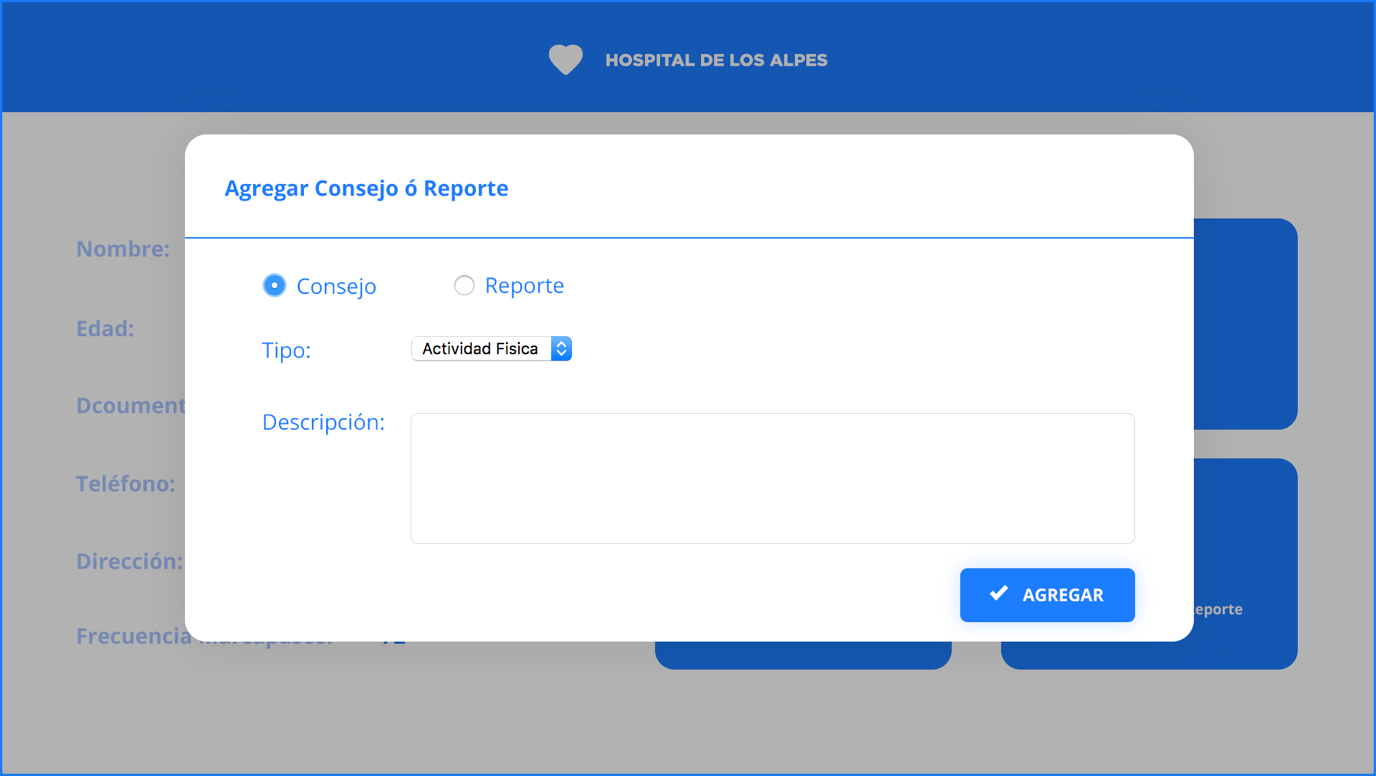
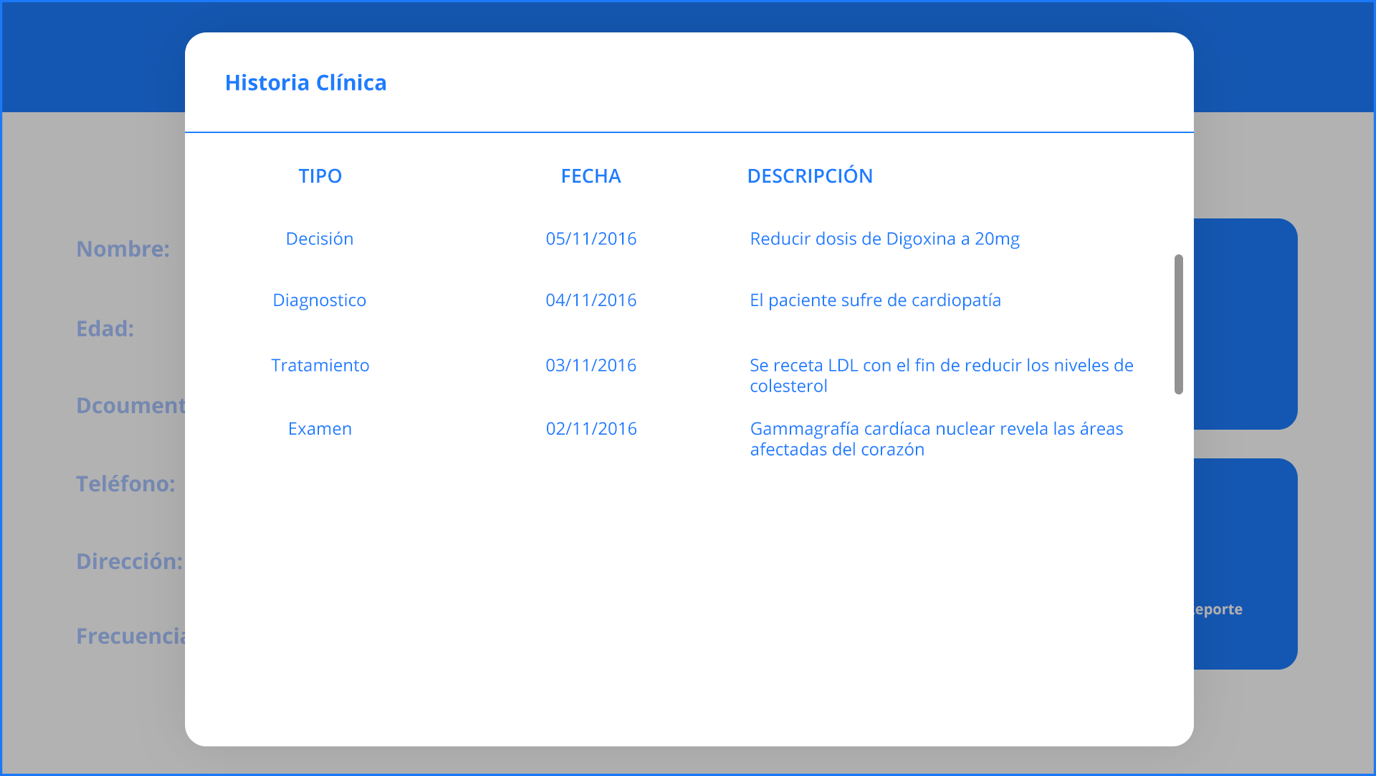
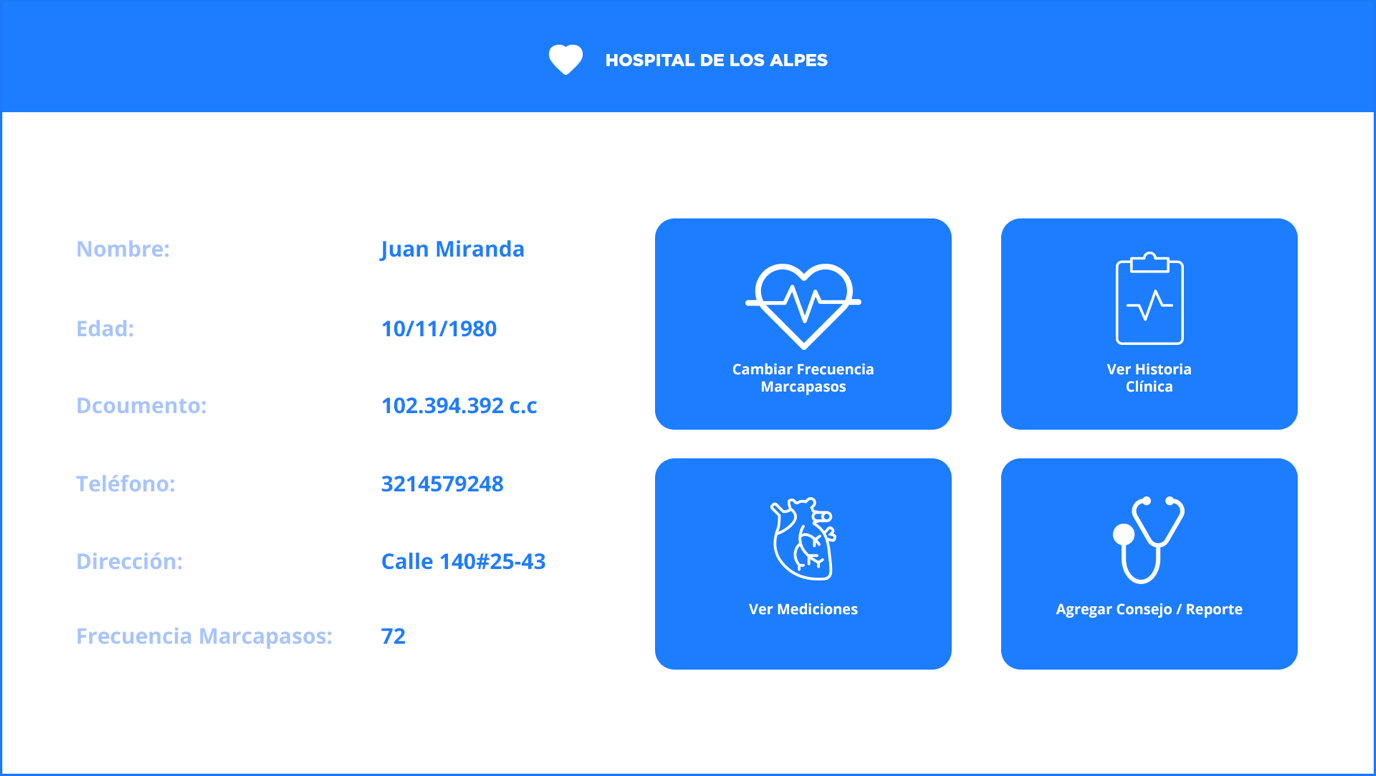
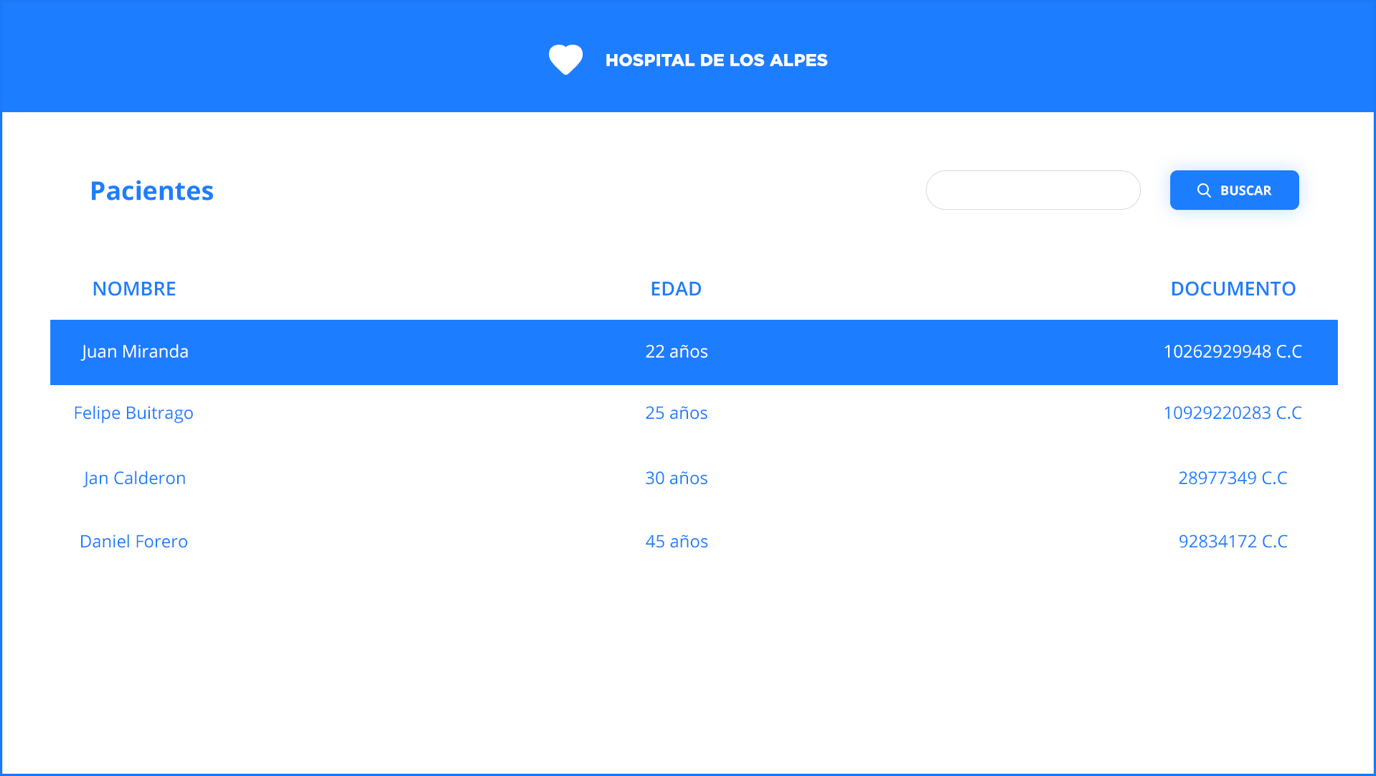
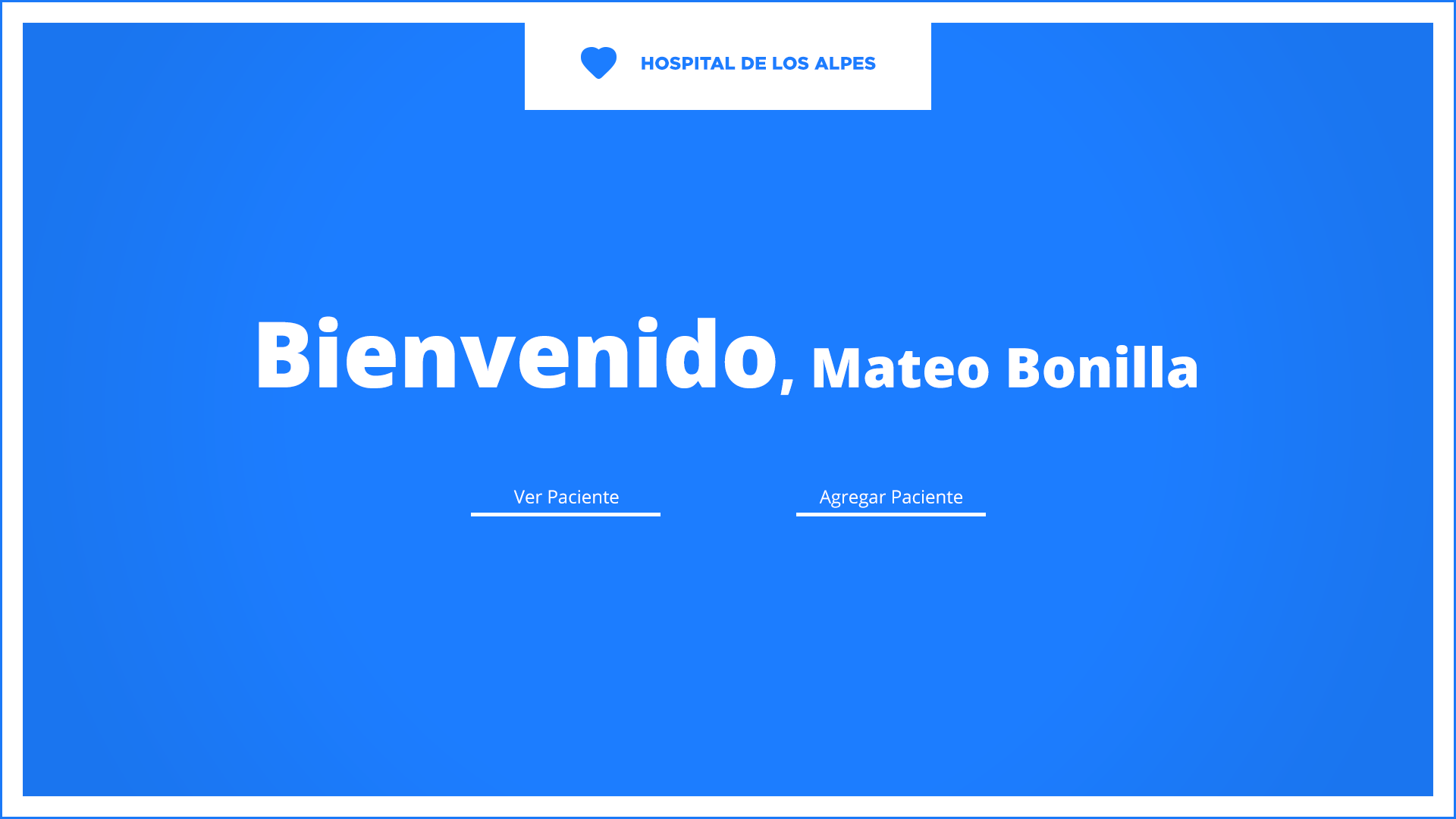


**Navegabilidad del usuario:**



**Usabilidad:**

Usabilidad

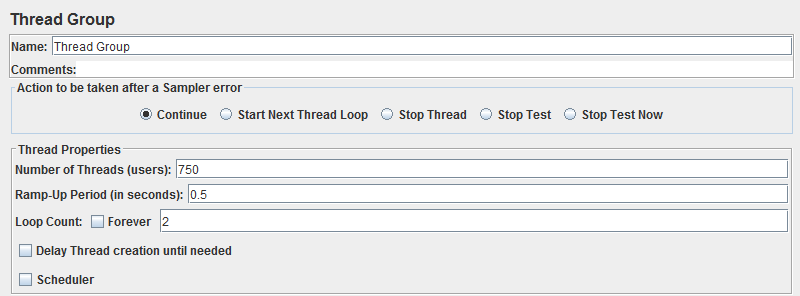


<https://goo.gl/forms/OfoNebNzMi87DXPy1>

Para el desarrollo del sistema se usó en un inicio una arquitectura multi-tier, esto con el fin de favorecer el desempeño y escalabilidad del sistema; en concreto usamos una arquitectura de dos tiers donde tenemos una tier de lógica (Backend) y otro donde se realiza la persistencia del sistema (Base de datos). Estas decisiones se hicieron basadas reflexionando en la distribución de cargas a través de los tiers, con el fin de mejorar los tiempos de ejecución del sistema aprovechando los recursos disponibles en cada uno de ellos, sumado a esto, se pensó en la escalabilidad del sistema ya que gracias a la arquitectura escogida los nuevos servicios que se quieran agregar podrán ser fácilmente adicionados sin alterar la lógica de los actuales.

De igual modo, hicimos uso de Jax RS para la realización del sistema con el fin de tener un completo control sobre los servicios REST; dado que, Jax-RS nos permite que la implementación del servicio REST que será consumido por el cliente sea sencilla de hacer gracias a las anotaciones implementadas dentro de java, además de permitir la escalabilidad del servidor sin afectar los servicios del REST, esto debido a que Jax-RS se encarga de “consumir” o generar los HTTP request y traducirlos para que el servidor no tenga que entender la lógica de los mismos.

Para las pruebas se usó una configuración de Jmeter en cluster. Desde 2 computadores diferentes se coordinaban pruebas como se muestran a continuación:

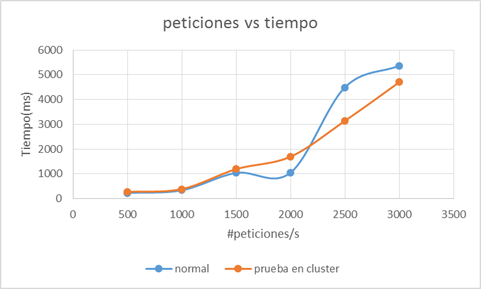


Se usaron 750 threads que se disparaban en medio segundo con una iteración. Esto hacia que cada uno de los threads creados en cada uno de los 2 computadores fuera reutilizado, para tener un total de 1500 peticiones en un segundo por cada computador. Al haber 2 computadores en el cluster, en total se probaba el programa con un total de 3000 peticiones por segundo.

Las pruebas en cluster permiten que no se gasten innecesariamente recursos de cada computador y hacen que el GUI de Jmetter no se sobrecargue.

Inicialmente se hizo un cargue de pacientes y luego se hicieron peticiones GET sobre estos pacientes.

También se hicieron las con una única máquina que enviaba las peticiones y se obtuvieron los siguientes resultados:

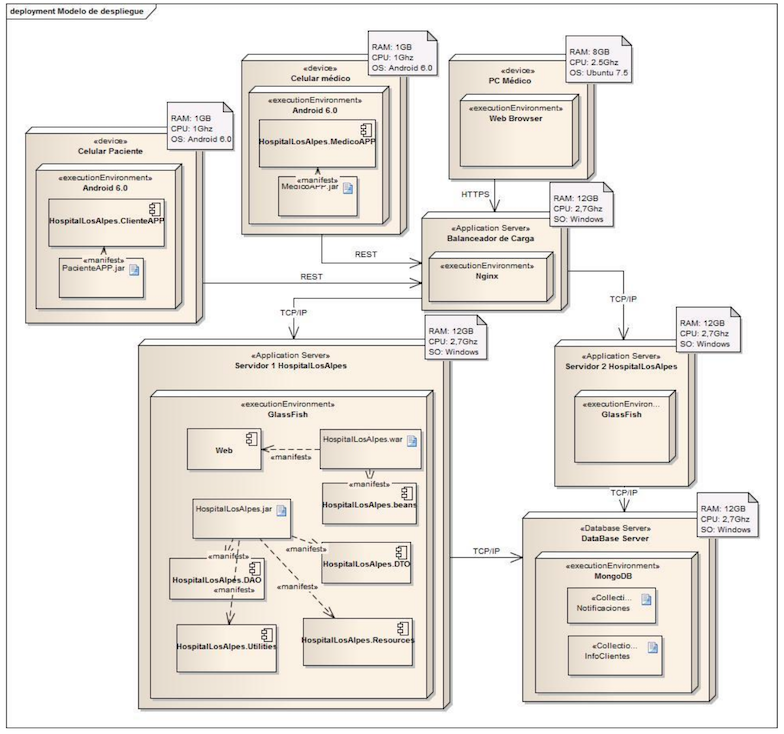


**Gráfica 2. GET pacientes**

Como puede observarse en la imagen, el tiempo de respuesta es muy similar para ambos tipos de pruebas, ya que estos dependen es del servidor.

Los tiempos de respuesta para las 3000 peticiones llega a ser casi de 5 segundos, un tiempo bastante alto. Este tiempo puede deberse al computador en dónde estaba corriendo el backend, ya que no es propiamente un servidor, además, también se debe tener en cuenta el tiempo de respuesta de la base de datos, ya que no se encuentra ubicada en la misma ubicación.

Con el fin de mejorar la disponibilidad del sistema, se optó por implementar una arquitectura como la del siguiente diagrama:



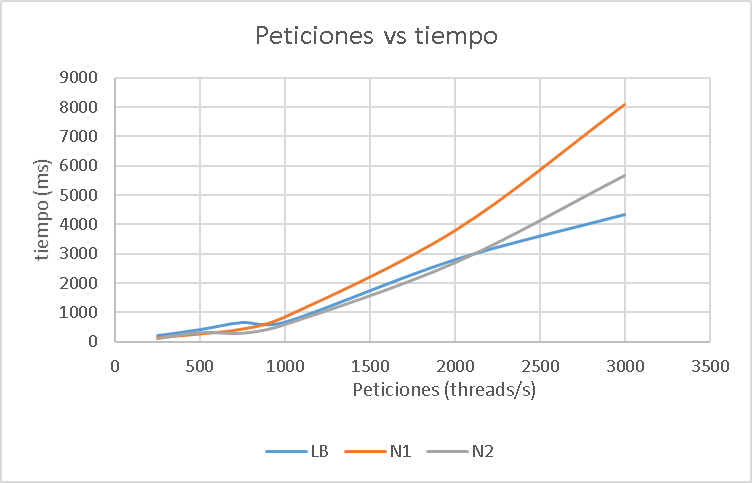
**Figura 1.** Arquitectura implementada

Despues de obtener los resultados anteriores se usaron 4 máquinas diferentes que ejecutaban peticiones sobre el balanceador de carga. El balanceador de carga se configuró usando el software de nginx e implementando la técnica de leastConn, la cual elige el servidor con menos carga en el momento para asignarle la tarea. Adicionalmente se usó la página de error por default del balanceador en caso que ninguno de los servidores de aplicaciones respondiera. El servidor de aplicaciones 1 tenía la aplicación corriendo sobre el IDE de Eclipse, mientras el nodo 2 tenía la aplicación desplegada en Netbeans.

Las pruebas se hicieron simulando peticiones POST de notificación sobre la plataforma. Inicialmente, se apuntó al balanceador de carga y luego a cada uno de los nodos directamente, obteniendo los siguientes resultados:

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Tiempo de respuesta (ms)** | | |
|  | **LB** | **N1** | **N2** |
| **250** | 217 | 147 | 107 |
| **500** | 421 | 273 | 313 |
| **750** | 667 | 441 | 294 |
| **1000** | 672 | 853 | 586 |
| **2000** | 2794 | 3791 | 2691 |
| **3000** | 4330 | 8090 | 5659 |

**Tabla 1.** Tiempos de respuesta método POST



**Figura 2.** Gráfica de los resultados de tiempos

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Porcentaje de error (%)** | | |
|  | **LB** | **N1** | **N2** |
| **250** | 0% | 0% | 0% |
| **500** | 0% | 0% | 0% |
| **750** | 0% | 0% | 0% |
| **1000** | 0% | 0% | 0% |
| **2000** | 4,35% | 7,93% | 0% |
| **3000** | 8,33% | 14% | 0% |

**Tabla 2.** Porcentajes de error

Como se puede observar en la Figura 2, los tiempos obtenidos con el balanceador son similares a los tiempos obtenidos apuntándole directamente al nodo 2, pero a partir de las 2200 peticiones por segundo, se observa que el balanceador empieza a obtener mejores tiempos de respuesta.

Adicionalmente se observa tanto en la Tabla 1 de tiempos como en la Tabla 2 de porcentajes de error, que el nodo 1 de la solución no tiene el mismo desempeño que el nodo 2 ya que este llega incluso a presentar errores cuando se envían 2000 peticiones, cosa que no sucede con el nodo 1. Sin embargo, al implementar el balanceador, el porcentaje de error asociado al nodo1 se disminuye casi a la mitad.

Con respecto a la entrega anterior, se tenían los siguientes tiempos de respuesta y porcentajes de error:

|  |  |  |
| --- | --- | --- |
|  | **Tiempo de respuesta (ms)** | **Porcentaje de error (%)** |
| **250** | 146 | 0% |
| **500** | 317 | 0% |
| **750** | 378 | 0% |
| **1000** | 735 | 0% |
| **2000** | 3248 | 0% |
| **3000** | 6886 | 0% |

**Tabla 3.** Valores experimento 1

Se observa que los valores son similares a los obtenidos apuntando únicamente a un nodo.

**Persistencia:**

Se decidió escoger una base de datos NO relacional como lo es MongoDB, ya que podemos hacer consultas Ad hoc, es decir, MongoDB soporta la búsqueda por campos, consultas de rangos y expresiones regulares. Las consultas a su vez pueden devolver un campo específico del documento pero también puede ser una función [JavaScript](https://es.wikipedia.org/wiki/JavaScript) definida previamente, lo anterior lo usamos para realizar consultas sin tener que introducir todos los atributos que componen la tabla, evitando campos vacíos y realizando de manera rápida la consulta deseada. En casos de emergencia es algo crucial ya que se pierde tiempo llenando datos inútiles cuando se necesita generar una solicitud inmediata a un problema que surge de improviso.

**Arquitectura nueva:**

En cuanto a la arquitectura, se le aplico otro nodo de base de datos Mongo DB que simplemente replica lo que se guarda en el nodo principal, lo anterior para garantizar la integridad de los datos. ( el proceso de replicación no afecta al programa base) y para acceder a dichas bases de datos es necesario realizar una autenticación previa.

