UNIVERSIDAD DE LOS ANDES FACULTAD DE INGENIERÍA

DEPARTAMENTO DE INGENIERÍA DE SISTEMAS Y COMPUTACIÓN

SISTEMA DE ALERTA TEMPRANA DE TSUNAMIS (SATT)

REPORTE ENTREGA PARCIAL EXPERIMENTO 2

Equipo de Trabajo:

Grupo 5

Integrantes:

Andrés Camilo Zuleta Calderón ac.zuleta10@uniandes.edu.co

Sergio Miguel Madera sm.madera10[@uniandes.edu.co](mailto:and-gome@uniandes.edu.co)

Camilo Carrillo ca.carrillo2369@uniandes.edu.co

Tabla de contenido

[RESUMEN EJECUTIVO 3](#_Toc451463394)

[ARQUITECTURA SELECCIONADA Y JUSTIFICACIÓN 3](#_Toc451463395)

[**Decisiones de arquitectura** 4](#_Toc451463396)

[Desempeño 4](#_Toc451463397)

[Escalabilidad 4](#_Toc451463398)

[Disponibilidad 4](#_Toc451463399)

[Seguridad 5](#_Toc451463400)

[Usabilidad 6](#_Toc451463401)

[Modificabilidad 6](#_Toc451463402)

[DEFINICIÓN DE LINEAMIENTOS DE CÓDIGO 6](#_Toc451463403)

[ANÁLISIS DE MÉTRICAS UTILIZANDO SONAR-QUBE 7](#_Toc451463404)

[CRÍTICA COOLABORATIVA 11](#_Toc451463405)

[RESULTADOS CRÍTICA COOLABORATIVA 13](#_Toc451463406)

[Referencias 13](#_Toc451463407)

# RESUMEN EJECUTIVO

Dentro del presente documento se encuentra el trabajo realizado por el grupo de trabajo número cinco para la entrega preliminar del primer experimento correspondiente al curso de Arquitectura y diseño de software. Este documento incluye un análisis de las decisiones tomadas en cuanto a la arquitectura y la justificación de estas, tomando en cuenta los requerimientos no funcionales planteados para esta entrega del experimento. En adición, se describe la planeación, el desarrollo y la ejecución de los servicios y pruebas de los atributos de calidad realizados sobre el sistema SATT construido.

# ARQUITECTURA SELECCIONADA Y JUSTIFICACIÓN

La arquitectura seleccionada para la realización del proyecto es la arquitectura basada en servicios REST de JAX-RS. Dicha decisión fue tomada inicialmente debido a la facilidad de manejo que esta otorga y por el potencial que esta le otorga al equipo desarrollador para desarrollar, valga la redundancia, de manera eficaz y ágil. Adicionalmente esta arquitectura cuenta con los beneficios que REST tiene para ofrecer y su separación por capaz permite un fácil manejo de sus distintos componentes, facilitando así su mantenimiento en el caso de que se deseara una longeva esperanza de vida para la aplicación y además su fácil modificación en caso de ser requerida.

Ahora, esta arquitectura resulta también beneficiosa para los objetivos del experimento, es decir, el cumplimiento de los atributos de calidad de desempeño y escalabilidad. Esto podemos deducirlo a partir de los resultados que obtuvimos durante el laboratorio de desempeño y escalabilidad llevado a cabo durante el curso, donde se pudo ver que una aplicación REST soportaba de manera eficaz tanto el desempeño como la escalabilidad para un número significante de usuarios (más de diez mil). Por tanto a partir de esto se puede deducir que JAX-RS permite manejar de manera adecuada el número de solicitudes que se espera para la aplicación realizada durante el experimento (alrededor de cuatro mil).

Finalmente el grupo de trabajo decidió que es adecuado permanecer con esta arquitectura hasta al final debido a su gran versatilidad y facilidad para acoplarse a las distintas tecnologías para satisfacer los distintos escenarios de calidad. Los cuales hasta ahora han sido cumplidos de forma efectiva tal y como se buscaba en los objetivos del curso.

# **Decisiones de arquitectura**

## Desempeño

Las decisiones tomadas con el grupo para satisfacer los escenarios de calidad propuestos por el enunciado que conciernen al atributo de calidad de desempeño son las siguientes.

En primer lugar se propone una arquitectura a tres capas, siendo estas la capa de presentación, la capa de servicios y finalmente la capa de lógica y persistencia. Lo anterior es un factor que contribuye significativamente al desempeño de la aplicación ya que al estar desacoplada la presentación de la aplicación de los servicios y de la lógica esto permite que la lógica y componentes que se encuentran ubicados en la capa de presentación no agreguen latencia ni afecten las solicitudes que son generadas por los usuarios.

En segundo lugar como manejador de base de datos de nuestra unidad de persistencia se utiliza MongoDB que al ser una base de datos no relacional es capaz de procesar de forma más rápida las operaciones del CRUD de los diferentes objetos y colecciones de objetos que son necesarios para el correcto funcionamiento de la lógica de la aplicación (y en general presenta un desempeño notoriamente superior en cuanto a operaciones de lectura se trata) lo cual nos apoya a satisfacer de forma efectiva los escenarios de calidad asociados al desempeño.

## Escalabilidad

Las decisiones tomadas con el grupo para satisfacer los escenarios de calidad propuestos por el enunciado que conciernen al atributo de calidad de escalabilidad son las siguientes.

En primer lugar y como anteriormente se mencionó, el grupo propone una solución utilizando una arquitectura a tres capas, capas que ya anteriormente fueron especificadas. La ventaja que esto supone en aras de la escalabilidad de la aplicación es que al estar separada la capa de presentación de los servicios y la lógica es que podemos correr dichas capas como instancias diferentes de la aplicación, es decir, podemos tratar la interfaz gráfica y la lógica como si de dos aplicaciones diferentes se tratasen y por tanto esto nos permite la posibilidad de no tener que subir a una plataforma cloud (como heroku ó AWS) toda la aplicación dentro de un solo paquete, lo cual nos generaría problemas de mantenibilidad del código a futuro y además problemas de la escalabilidad dado el alto acoplamiento entre las capas (debido a que para tener una instancia de una aplicación hecha en JEE es necesario empaquetar toda la aplicación dentro de una sola capa y generar un main que la ejecute). De esta forma al subir solamente la capa de lógica y la capa de servicios a una plataforma cloud podemos utilizar fácilmente las funcionalidades ofrecidas por esta, su alta disponibilidad, alto rendimiento y por supuesto, alta flexibilidad (lo cual es incluso mejor que la escalabilidad) y dejar que la plataforma administre dichos recursos por nosotros, liberando así al equipo de desarrolladores de ser los encargados de supervisar dichas tareas y solamente concentrándose en mantener la capa de presentación la cual estaría haciendo siempre peticiones de servicios REST a la instancia de la aplicación que se encuentra corriendo en la nube.

En segundo lugar en esta ocasión debemos volver a mencionar el manejador de base de datos utilizado por el grupo, MongoDb, pues este manejador ofrece dentro las posibilidades de sus configuraciones la capacidad de crear clusters de instancias de bases de datos lo cual nos permite ofrecer la capacidad de escalar con facilidad ya que los clusters de bases de datos permiten un procesamiento más rápido de los datos que son allí guardados y optimiza las operaciones de CRUD que son realizadas en los objetos de las colecciones que allí se encuentran.

## Disponibilidad

Las decisiones tomadas con el grupo para satisfacer los escenarios de calidad propuestos por el enunciado que conciernen al atributo de calidad de disponibilidad son las siguientes.

En primer lugar para asegurar la alta disponibilidad de los servidores encargados de resolver las solicitudes generadas por los usuarios de la aplicación fue el uso de un balanceador de carga que distribuyera de forma equitativa las solicitudes. Dicho balanceador de carga es un factor decisivo para poder satisfacer este atributo de calidad. Lo anterior debido a que la inclusión de dicho balanceador dentro de nuestra arquitectura nos permite tener varias instancias de nuestra aplicación corriendo en distintos servidores, esto nos permite tener una mayor confianza de que nuestro sistema será de alta disponibilidad puesto que al tener una mayor cantidad de servidores disponibles para solucionar solicitudes tendremos la confianza de que en caso de que algún servidor llegue a estar indisponible, ya sea por motivos técnicos, de sobrecarga del servidor por culpa de la alta demanda en las solicitudes y por fallos humanos, tendremos otra instancia de nuestra aplicación que se encargará de solucionar las peticiones que fueron asignadas a otro servidor. Todo lo anterior claramente soportado por el balanceador de carga el cual fue configurado por el grupo para que, en primer lugar las peticiones fueran direccionadas al servidor que poseía la mayor cantidad de recursos para resolver peticiones (directiva least\_conn de nginx), guardará en cache las sesiones de los usuarios para resolver más rápidamente sus solicitudes (sticky\_cookies) y finalmente en caso de que un servidor quedara inhabilitado, al momento de la recuperación de este no le llegaran solicitudes de forma inmediata con el fin de evitar una posible caída de dicho servidor por culpa de una saturación por alta demanda en la cantidad de solicitudes recolectadas ( directiva slow\_time). Con todo lo anteriormente dicho es la forma en la que garantizamos la alta disponibilidad de nuestro sistema a nivel de los servidores encargados de resolver las peticiones de los usuarios.

Adicionalmente la inclusión del balanceador de carga nos permite escalar de forma horizontal nuestra aplicación pues como se mencionó este distribuye de forma óptima las peticiones entre los servidores, o sea que entre más servidores tengamos obtendremos una mejora sustancial en la escalabilidad del sistema.

En segundo lugar volveremos a mencionar el uso del manejador de base de datos MongoDb quien es esta ocasión al ofrecer con facilidad la capacidad de configurar conjuntos de réplicas de las instancias de la base de datos brinda la posibilidad de tener una instancia de base de datos en cada servidor que esté resolviendo solicitudes, dichas instancias se sincronizan cada cierto tiempo (elegido por nosotros, en este caso 500ms) asegurando así la integridad y consistencia de los datos en cada una de las instancias de la aplicación y adicionalmente al ofrecer MongoDB un sistema de arbitraje en el cual existe una instancia principal que se encarga de coordinar a todas las instancias secundarias para que no se presente inconsistencia de datos y en el momento en que dicha instancia principal llegue a caer sea capaz de nombrar a una instancia secundaria como la principal de forma provisional mientras la anterior instancia principal se recupera nos garantiza que no solamente nuestro sistema tendrá una alta disponibilidad a nivel de resolución de las peticiones y de los servidores sino también a nivel del manejo y almacenamiento de los datos.

## Seguridad

Las decisiones tomadas con el grupo para satisfacer los escenarios de calidad propuestos por el enunciado que conciernen al atributo de calidad de disponibilidad son las siguientes.

En primer lugar para la seguridad el grupo de trabajo decidió utilizar el protocolo de seguridad SSL (**capa de puertos seguros) [5]** el cual es necesario para poder utilizar el protocolo de comunicación de https. El anterior protocolo propuesto fue integrado dentro de la aplicación utilizando el framework de shiro [3] y la utilización de este para servidores JAX-RS, como en el caso de nuestra aplicación, garantiza de forma efectiva tanto la integridad como la confidencialidad de de los datos. Lo anterior es confirmado no solo por la investigación que realizó nuestro grupo de trabajo si no que también es afirmado por entidades confiables y expertas en el tema como los es la IBM quienes mencionan que utilizando SSL dentro de servidores JAX-RS “Ha definido una conexión segura entre el cliente y el servidor de destino utilizando SSL para habilitar la integridad y la confidencialidad de la comunicación entre la aplicación de JAX-RS y el cliente.” [1]. Por tanto gracias a esto podemos afirmar que estamos asegurando tanto la integridad como la confidencialidad de los datos según las expectativas de experimento 2.

En segundo lugar el grupo de trabajo para garantizar tanto la autenticación como la autorización a los servicios de la aplicación utilizó el framework de shiro integrado al servidor de stormpath el cual por medio del manejo de roles y tokens para las sesiones permite de forma fácil un mecanismo de autenticación básica que integrado de forma gráfica con angular nos permitió crear un servicio básico de login con el cual garantizamos que cada usuario ingrese únicamente a los servicios a los cual tiene derecho dependiendo de sus roles.

## Usabilidad

Para la toma de decisiones en cuánto a la usabilidad el primer factor que tomamos en cuenta es que al tratarse de una aplicación empresarial que es utilizada por una organización donde la organización de la información y su representación de forma precisa es el principal core de negocio, entonces decidimos que lo más importante era la forma en que la información es mostrada. Por tanto optamos por una interfaz sobría a nivel colores y con tamaños de letras grandes para facilitar su visualización según lo aprendido de Ley de Fitts durante el curso. Con las diferentes funcionalidades posibles siendo accesibles desde la barra de tareas que se ubica en la parte de arriba, en todo el centro del navegador (táctica de usabilidad también aprendida durante el curso y que fue puesta en práctica para nuestro proyecto) y finalmente implementamos una opción de filtro para que el experto pudiera organizar la información de acuerdo a sus necesidades. Todo lo anterior para la versión web de la aplicación SATT.

Para la aplicación móvil se utilizaron las tácticas de usabilidad para móviles vistas en clase, entre ellas, colocar las funcionalidades en el centro de la pantalla, lugar donde son más visibles, no colocar publicidad innecesaria y colocar filtros de búsqueda. Adicionalmente todos los anteriores criterios fueron fortificados al añadir la paginación sobre el servidor para que se vayan obteniendo una serie de resultados que logran ser visualizados de forma efectiva en el navegador para no disminuir la productividad del usuario al utilizar la aplicación web/móvil.

Finalmente se implementaron las mejoras propuestas por los resultados obtenidos de la encuesta de crítica colaborativa que fue realizada a diferentes usuarios (conocidos del grupo de trabajo) a quienes se les explicó el contexto bajo el cual se usaría la aplicación y se les pidió que respondieran a dicha encuesta como si estuvieran ocupando el lugar, en primer lugar, del administrador, y en segundo lugar, del experto de la DGR.

## Modificabilidad

En cuanto a la modificabilidad del código desde el comienzo del proyecto se pensó en que el manejo del proyecto a nivel de capas (en este caso 3) nos brinda una gran ventaja para este atributo de calidad puesto que permite una alta cohesión del proyecto al estar desacopladas las distintas capas a las cuales se les encargan las diferentes responsabilidades que son necesarias para que la aplicación solucione el problema. Adicionalmente se usó la táctica de encapsulamiento a nivel de interfaces vista en clase la cual también nos permite obtener un código más modificable al permitirnos tener distintas implementaciones de los distintos servicios expuestos dependiendo de las necesidades en las cuales se encuentren los distintos stakeholders de la aplicación.

Finalmente al aplicar los distintos lineamientos de código, que serán expuestos luego en el resto del documento, junto con las métricas ofrecidas por SonarQbe logramos generar un código a base de buenas prácticas de programación que cumple con los estándares a nivel mundial y que permite al grupo de desarrolladores un mayor entendimiento del código y mantenimiento del mismo.

# DEFINICIÓN DE LINEAMIENTOS DE CÓDIGO

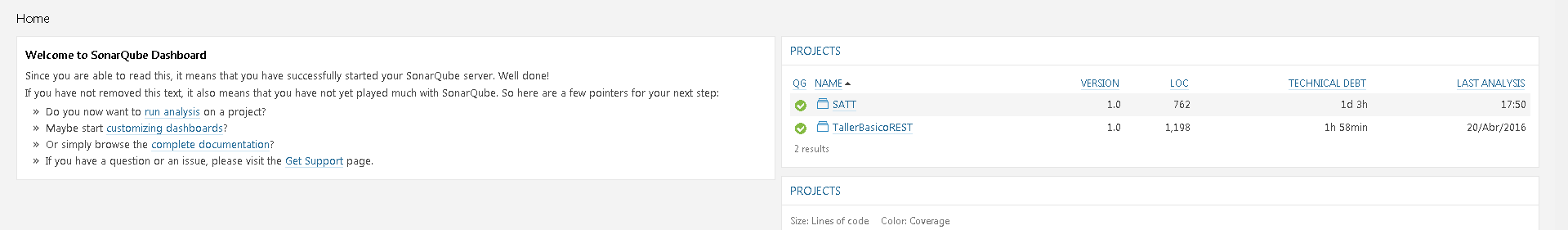
Los distintos lineamientos que el grupo de trabajo utilizo para optimizar la modificabilidad del código y asegurarse de la facilidad del mantenimiento de este gracias a las buenas prácticas de programación fueron:

1. Añadir comentarios explicando el porqué de los argumentos o la ausencia de estos para el constructor de una clase. Lo anterior para clarificar a los desarrolladores para qué sirve cada argumento que entra como parámetro en un constructor y se puedan dar una mejor idea de lo que hace cada método, en este caso, los métodos constructores y en el caso de que no posean argumentos una explicación breve y concisa que esclarezca porqué se utiliza un constructor vacío.
2. Cambiar el especificador de tipo en el constructor de un arrayList de <tipo> por <>. Lo anterior debido a que el operador diamante, es decir ‘<>’, añade genericidad al código y por tanto lo hace más mantenible.
3. No utilizar como retornos o parámetros de métodos, implementaciones de objetos o estructuras de datos muy específicas, por ejemplo, ArrayList. Si más bien utilizar las interfaces genéricas en las cuales dichas implementaciones están basadas, por ejemplo, en el caso del ArrayList en vez de colocar ArrayList colocamos List. Lo anterior debido a que pueden existir muchas implementaciones de una interfaz, como es el caso de ArrayList para List y al declarar el retorno o la variable como un tipo genérico nos brinda versatilidad para programar y permite cambiar de forma más fácil el código.
4. Crear un registro de las excepciones que pueden ocurrir para que no se pierda el trazo de estas y podamos identificar la fuente del error.
5. Utilizar la anotación @FunctionalInterface para las interfaces creadas para los distintos EJB´s (en el caso de usarse interfaces). Lo anterior, según el estándar JEE para declarar que efectivamente la interfaz utilizada cumple una funcionalidad con relación a las implementaciones que se hagan de esta misma.
6. En vez de utilizar utilidades de sistema para notificación de errores o advertencias es mejor utilizar utilidades de instancias que deriven de la clase logger. Lo anterior debido a que la clase Logger nos permite añadir más información a las notificaciones que nos llegan, por ejemplo, el objeto que generó el error o el nivel de “peligro” que representa el mensaje de información que nos llega en pantalla, el cual no podemos configurar con una simple impresión de consola pero sí podemos definir utilizando los niveles implementados por la clase logger. Por ejemplo, podemos decir que un mensaje de información tiene un nivel “Severo” ó “Grave” ó “Normal” y con eso determinar qué tan grave es para nuestro código esa información que nos imprima la consola.
7. Unir las condiciones de los condicionales if y while utilizando conjunciones lógicas para evitar la creación innecesaria de condicionales anidados que pueden ser integrados en un solo if/while.
8. Utilizar excepciones propias para describir de una manera más efectiva y que transmita una mayor cantidad de información del error concreto a los desarrolladores en vez de usar excepciones genéricas que solo dicen que sucedió un error.
9. Finalmente no usar paréntesis, asignaciones y comentarios innecesarios. Las razones de esto son obvias, pueden llegar a confundir al desarrollador y confundirlo de las cosas que hace el código pues cada variable y asignación que se haga debe tener una razón de ser y el uso innecesario de alguno de los artefactos anteriormente mencionados dificulta la lectura y entendimiento del código además de, en el caso de las asignaciones, puede llegar a hacerlo más lento en tiempo de ejecución.

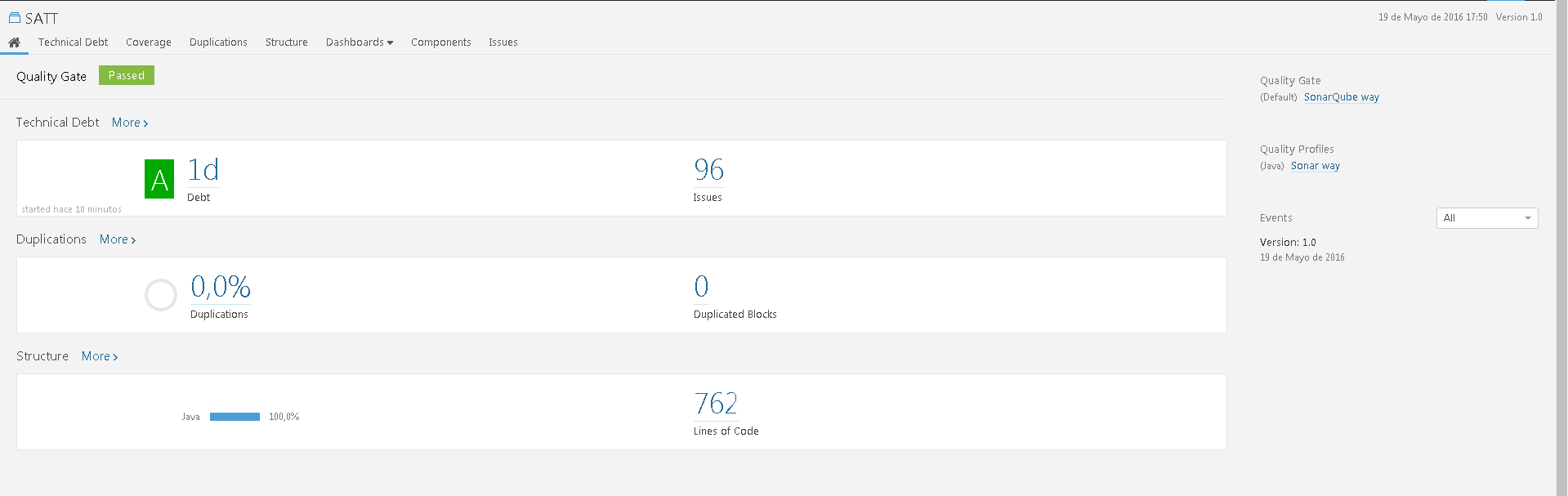
# ANÁLISIS DE MÉTRICAS UTILIZANDO SONAR-QUBE

En primer lugar se colocó el código fuente dentro de la sección de proyectos del sonar-runner con el fin de poder obtener un análisis del código por parte de la aplicación sonar-qube. Los resultados obtenidos de forma preliminar fueron los siguientes:

**Resultados obtenidos del código utilizado hasta el experimento 2:**

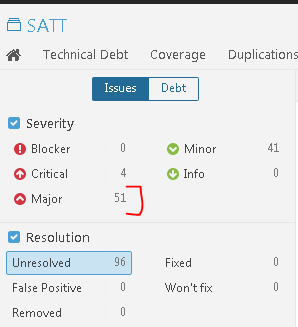


En la siguiente imagen se puede observar la cantidad de issues que poseía el proyecto inicialmente y la deuda técnica que poseía el proyecto hasta la realización del experimento 2:



Como se puede observar el proyecto realizado hasta el momento de la presentación del experimento 2 poseía una cantidad de 96 issues y una deuda técnica de 1 día con la cual el equipo de trabajo debía lidiar.

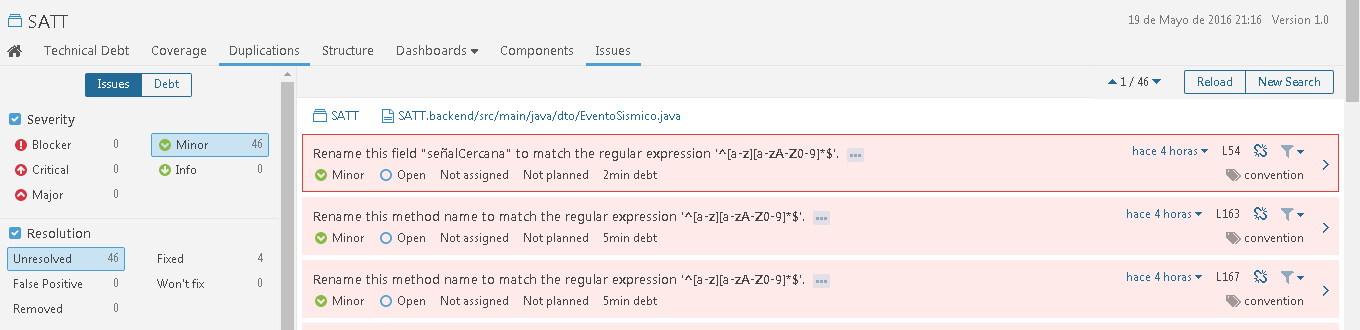
Ahora, al observar más a fondo observamos que la distribución y clasificación de los issues que poseía el proyecto estaba organizada de la siguiente forma:



Como se puede observar poseíamos una cantidad de 51 issues de tipo major, algo bastante grave y que reflejaba que el código que habíamos realizado estaba lleno de malas prácticas de programación, sobretodo al considerar que la cantidad de issues de tipo major superaba en una cantidad de 10 a los issues de tipo minor y que además presentábamos 4 issues de tipo crítico que podían afectar de forma severa la modificabilidad del código.

Por tal motivo se comenzaron a resolver los distintos issues que mostraba Sonar Qube utilizando y siguiendo al pie de la letra todos y cada uno de los lineamientos que fueron presentados anteriormente con el fin de lograr un código implementado bajo una serie de lineamientos óptimos de código que fuesen regidos por las buenas prácticas de programación y que lograsen un código cada vez más entendible para el equipo de desarrollo y que además fuese de fácil mantenimiento.

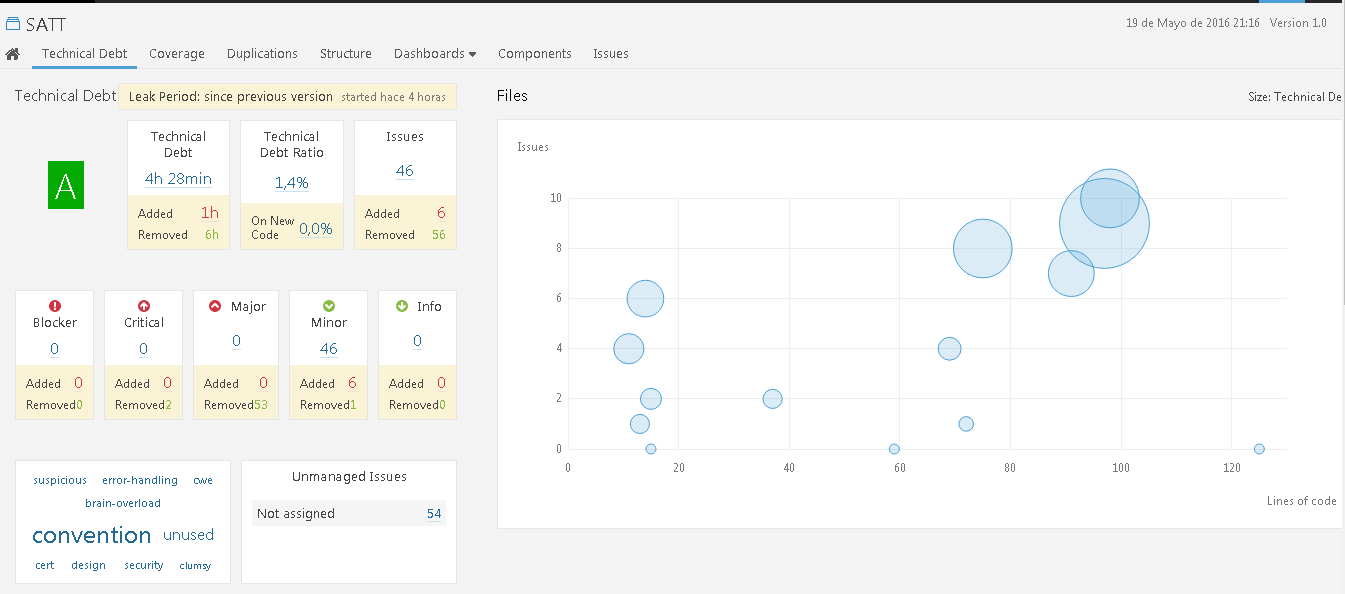
Siguiendo dichos lineamientos y con el apoyo de la herramienta Sonar Qube obtuvimos los siguientes resultados luego del trabajo de optimización y mejoramiento del código:



Como podemos observar en la siguiente imagen luego de aplicar las recomendaciones de Sonar Qube y seguir los lineamientos de código inicialmente planteados el grupo de trabajo logró eliminar por completo la presencia de issues de tipo major y de tipo crítico, dejando únicamente los issues de tipo minor debido a que estos no representan un problema muy grande para el mantenimiento del código y adicionalmente el tiempo que tomaría resolver estos issues tomaría mucho tiempo y cambiaría mucho el código. Es decir, el beneficio obtenido si se llegara a arreglar estos issues no sería suficiente para justificar el tiempo invertido para solucionarlos, por tanto estos se dejaron tal cual se observaron.

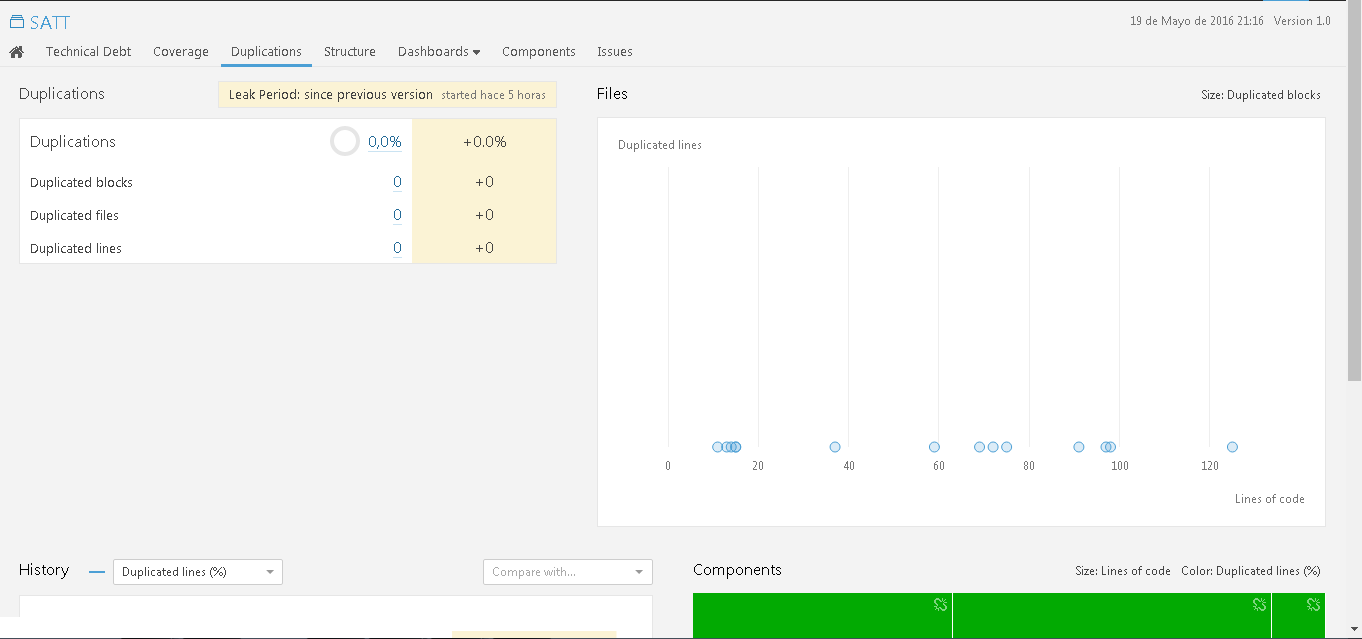


Finalmente luego de corregir el código estas fueron las métricas que fueron mostradas por el análisis realizado por Sonar Qube:

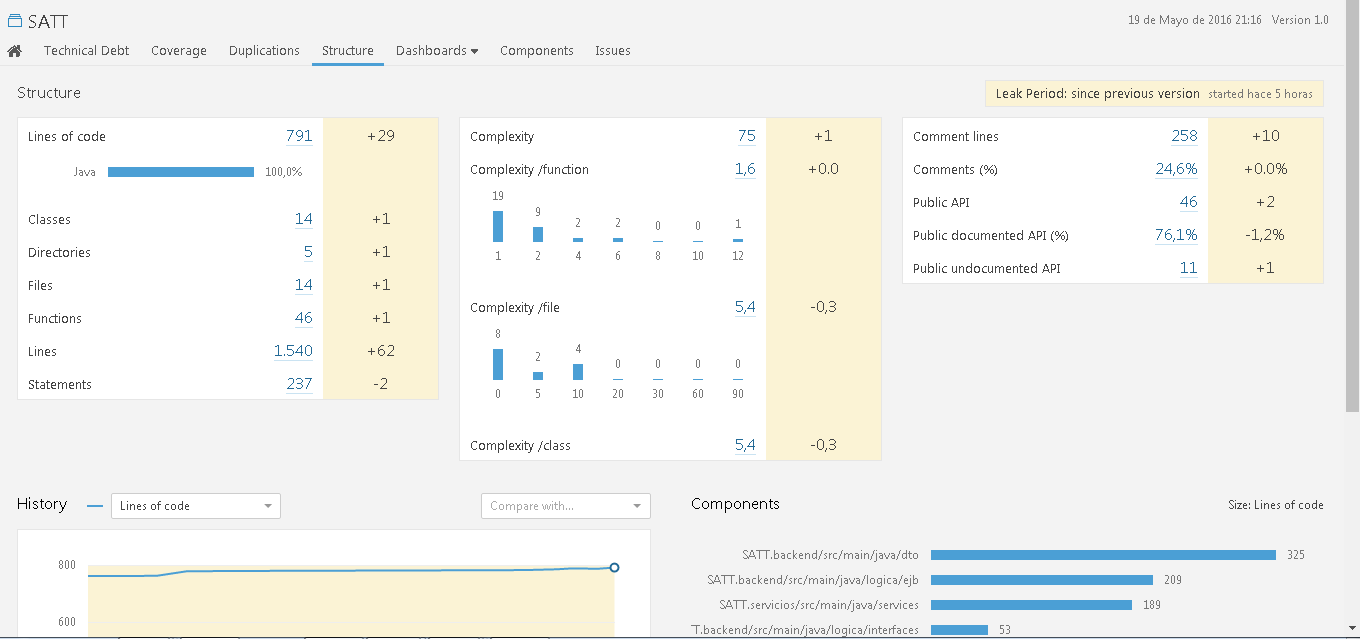


Como se puede observar en la imagen anteriormente presentada el grupo de trabajo logró disminuir la deuda técnica presentada por el proyecto de 1 día (la cual era la deuda técnica inicial del proyecto antes de corregir el código) a una deuda técnica de apenas 4 horas y 28 minutos lo cual es considerado por Sonar Qube como una deuda técnica decente y que no representa mayores repercusiones pata el mantenimiento del código en el futuro.

Adicionalmente y como se puede ver en la imagen de abajo, presentamos un porcentaje del 0% en cuanto a duplicaciones de código. Este porcentaje se mantuvo constante durante todas las versiones del proyecto puesto que incluso en la versión preliminar del proyecto que fue analizada usando Sonar Qube también había un porcentaje del 0% en cuanto a duplicación de código, por tanto en este aspecto siempre se mantuvo un producto hecho con calidad y buenas prácticas que no permitieron que existieran líneas de código repetidas.



Finalmente debemos mencionar que con respecto a la complejidad del código y a la cantidad de líneas de código obtenidas con relación a la primera versión del proyecto que fue analizada con Sonar Qube podemos mencionar de forma resumida lo siguiente: Las líneas de código aumentaron en 29, se implementó una clase nueva para manejo de excepciones personalizadas, la complejidad total del código aumentó en una unidad pero se redujo en -0,3 por archivos. Finalmente se aumentó el porcentaje de comentarios debido a que se agregaron 10 líneas nuevas para documentar partes del código que no contaban con documentación. Por tanto mejoramos la modificabilidad por este lado aunque se aumentó un poco la complejidad del código, aunque no mucho, por tanto esta diferencia es irrisoria.



Por tanto, teniendo en cuenta todos los argumentos anteriormente expuestos debemos mencionar que reducimos la cantidad de issues de 96 issues que era la cantidad inicial de estos a tan solo unos 46 issues a la vez que se eliminaron todos los issues de tipo major y críticos, mantuvimos un porcentaje del 0% de líneas de código duplicadas, se aumentaron las líneas de código en 29 líneas, se aumentó la cantidad de líneas de código dedicadas a documentación en 10 líneas y finalmente se aumentó en una unidad la complejidad del código. Por lo cual el grupo de trabajo considera como un gran éxito a nivel de modificabilidad del código. Por tanto podemos afirmar que gracias a Sonar Qube logramos obtener una versión superior del proyecto con relación a las versiones anteriormente presentadas.

## CRÍTICA COOLABORATIVA

La crítica coolaborativa realizada por el grupo de trabajo fue la siguiente:

Inicio Sesión:

* ¿Encontró fácilmente la funcionalidad de inicio de sesión? SÍ/NO
* ¿Recordaría su ubicación en próximos usos de la app? SÍ/NO
* ¿Fue intuitivo el uso de esta funcionalidad? SÍ/NO
* ¿Recordaría los pasos a seguir para poder usarla en próximos usos? SÍ/NO
* ¿Presentó errores al momento de usar esta funcionalidad? SÍ/NO
* ¿Qué problemas presentó al utilizar esta funcionalidad?

Ingresar señal de un sensor:

* ¿Encontró fácilmente la funcionalidad de ingresar señal de un sensor? SÍ/NO
* ¿Recordaría su ubicación en próximos usos de la app? SÍ/NO
* ¿Fue intuitivo el uso de esta funcionalidad? SÍ/NO
* ¿Recordaría los pasos a seguir para poder usarla en próximos usos? SÍ/NO
* ¿Presentó errores al momento de usar esta funcionalidad? SÍ/NO
* ¿Qué problemas presentó al utilizar esta funcionalidad?

Visualizar información de sensores:

* ¿Encontró fácilmente la funcionalidad de visualizar información de sensores? SÍ/NO
* ¿Recordaría su ubicación en próximos usos de la app? SÍ/NO
* ¿Fue intuitivo el uso de esta funcionalidad? SÍ/NO
* ¿Recordaría los pasos a seguir para poder usarla en próximos usos? SÍ/NO
* ¿Presentó errores al momento de usar esta funcionalidad? SÍ/NO
* ¿Qué problemas presentó al utilizar esta funcionalidad?

Enviar información de un evento sísmico:

* ¿Encontró fácilmente la funcionalidad de enviar información de un evento sísmico? SÍ/NO
* ¿Recordaría su ubicación en próximos usos de la app? SÍ/NO
* ¿Fue intuitivo el uso de esta funcionalidad? SÍ/NO
* ¿Recordaría los pasos a seguir para poder usarla en próximos usos? SÍ/NO
* ¿Presentó errores al momento de usar esta funcionalidad? SÍ/NO
* ¿Qué problemas presentó al utilizar esta funcionalidad?

Ver información histórica de reportes:

* ¿Encontró fácilmente la funcionalidad de ver la información histórica de los reportes? SÍ/NO
* ¿Recordaría su ubicación en próximos usos de la app? SÍ/NO
* ¿Fue intuitivo el uso de esta funcionalidad? SÍ/NO
* ¿Recordaría los pasos a seguir para poder usarla en próximos usos? SÍ/NO
* ¿Presentó errores al momento de usar esta funcionalidad? SÍ/NO
* ¿Qué problemas presentó al utilizar esta funcionalidad?

## RESULTADOS CRÍTICA COOLABORATIVA

A continuación se presentan los resultados obtenidos de la crítica coolaborativa que fue realizada. Hay que aclarar que dicha encuesta fue realizada a 10 personas distintas de un rango de edad entre 18 – 22 años y todos estudiantes de la facultad de ingeniería de la universidad de los Andes.

¿Encontró fácilmente la funcionalidad de ver la información histórica de los reportes?

¿Recordaría su ubicación en próximos usos de la app?

¿Fue intuitivo el uso de esta funcionalidad?

¿Recordaría los pasos a seguir para poder usarla en próximos usos?

¿Presentó errores al momento de usar esta funcionalidad? SÍ/NO

¿Qué problemas presentó al utilizar esta funcionalidad?

Los principales problemas manifestados por las personas encuestadas fueron:

* No había claridad en los valores de entrada que debían ingresar y la respuesta ingresada. Sin embargo esto se debe a la falta de comprensión en totalidad del contexto sobre el cual se encuentra el funcionamiento de la aplicación.
* A veces se presentaban problemas con el tipo de usuario a autenticar y a las opciones que tenían disponibles de acuerdo a sus roles.
* No había forma de regresar al home o pantalla de inicio principal de la aplicación.

Finalmente y con base a los resultados y comentarios obtenidos gracias a la realización de la encuesta de crítica colaborativa el grupo de trabajo decidió realizar una serie de pequeñas mejoras a la interfaz gráfica desplegada para la aplicación con el fin de satisfacer más eficazmente las necesidades de los usuarios.

# Referencias

[1] http://www.ibm.com/support/knowledgecenter/SSAW57\_8.0.0/com.ibm.websphere.nd.doc/info/ae/ae/twbs\_jaxrs\_impl\_securejaxrs\_clientssl.html?lang=es

[2] https://docs.mongodb.org/manual/tutorial/deploy-replica-set/

[3] http://shiro.apache.org/web.html

[4] <http://shiro.apache.org/static/current/apidocs/org/apache/shiro/web/filter/authz/SslFilter.html>

[5] https://es.wikipedia.org/wiki/Transport\_Layer\_Security