Universidad de los Andes

Arquitectura de software

Modificabilidad

Experimento 3

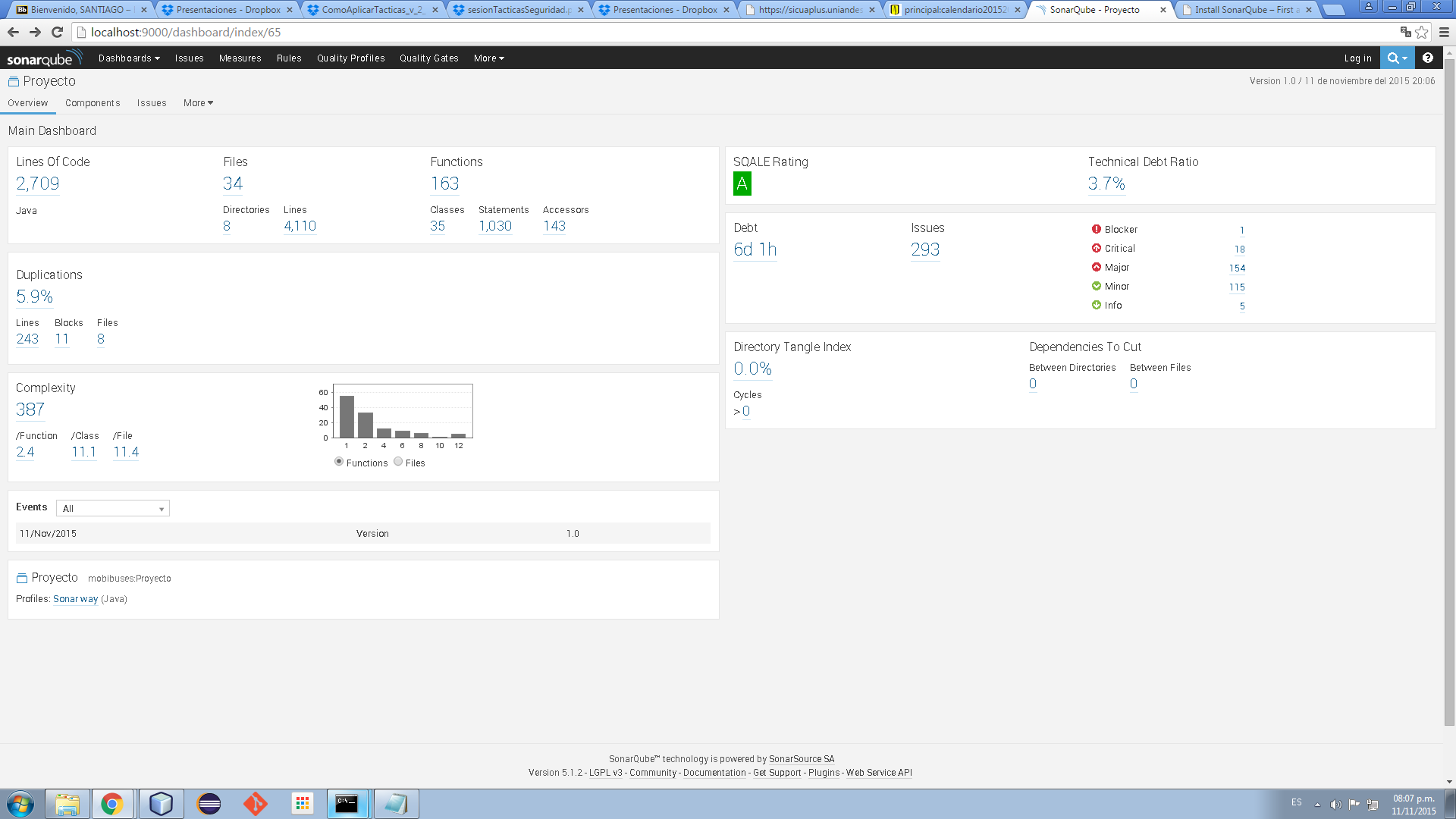
Entrega 1

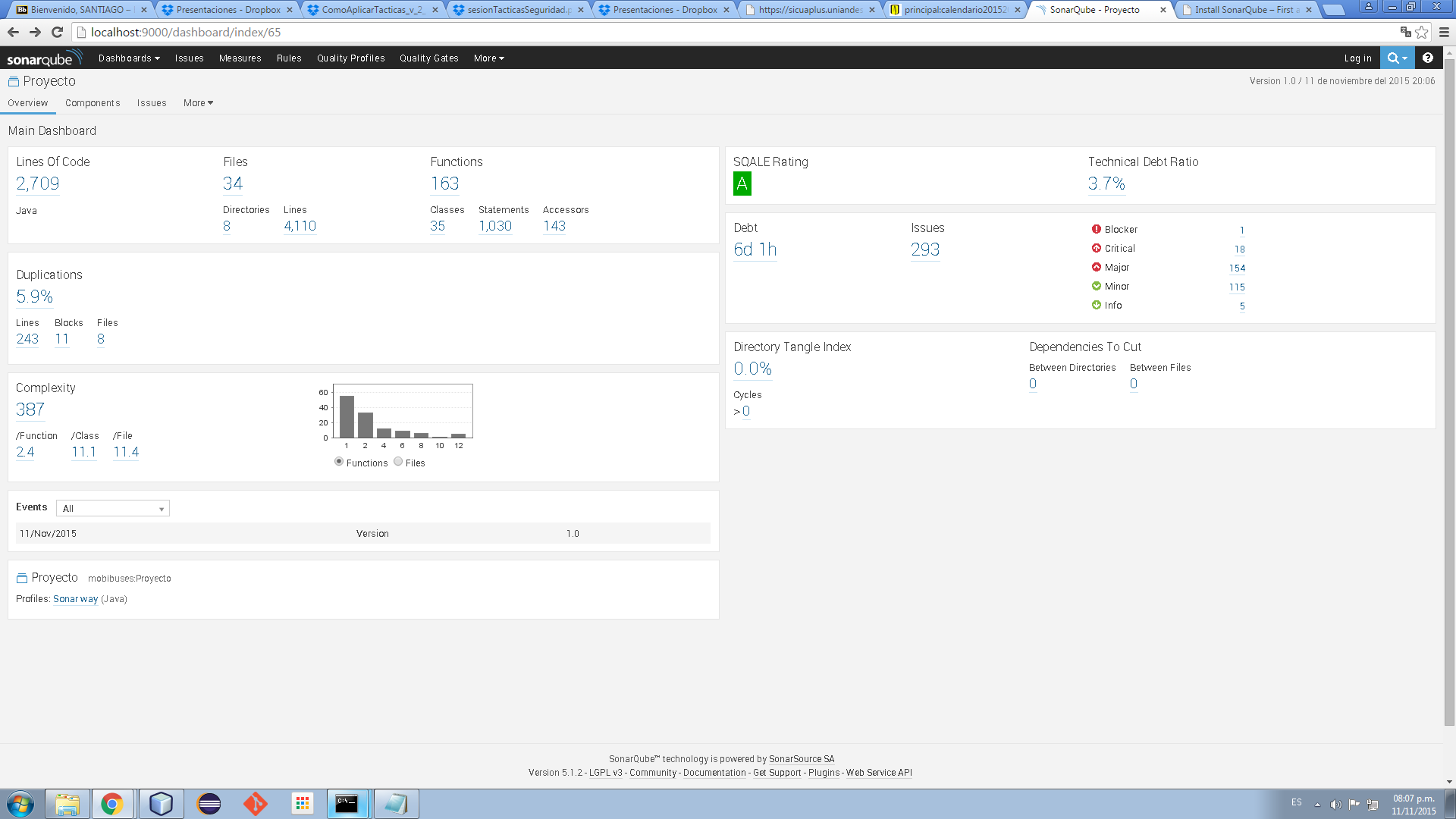
Royal Solutions

2015-2

Sonar

Con la ayuda de la herramienta de evaluación de código SonarCube se logró solucionar una serie de problemas que podrían impactar negativamente lo modificabilidad de la aplicación. A continuación presentamos la deuda técnica que inicialmente presentó la herramienta:

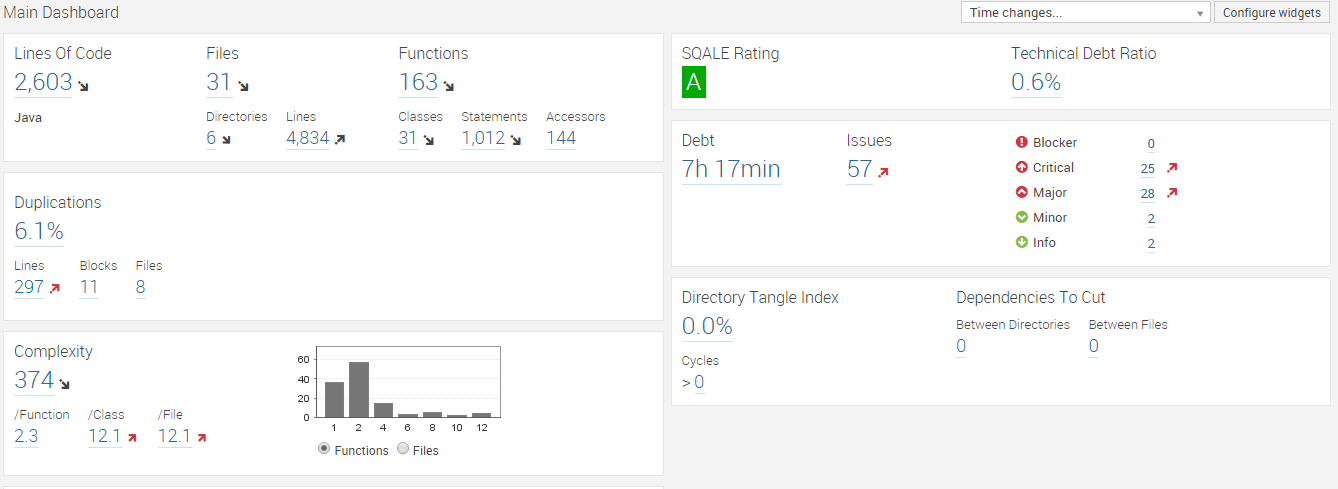
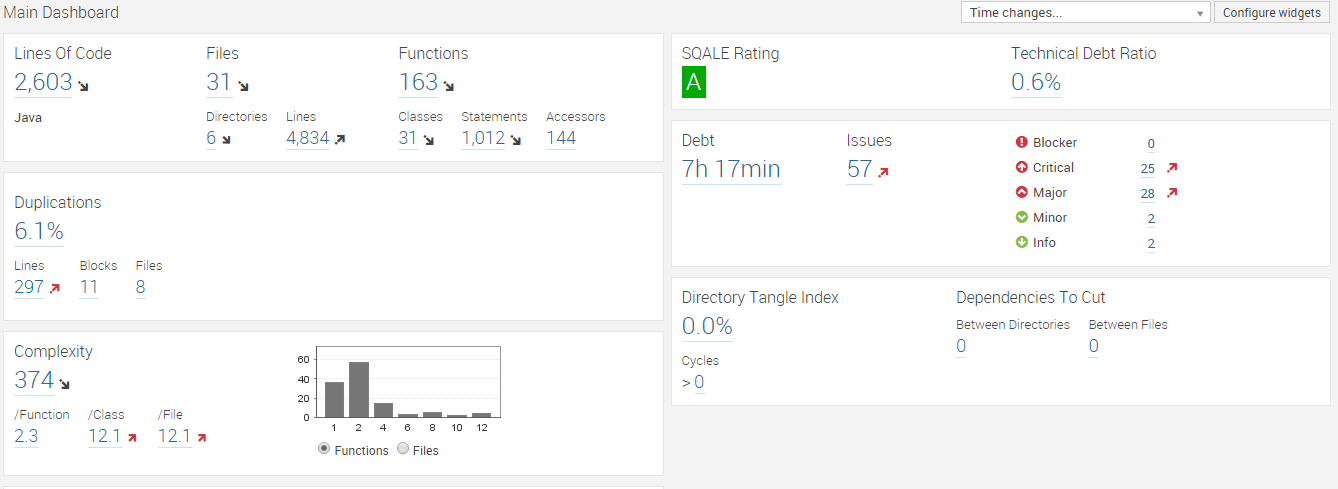




Se muestra que aunque hay varios problemas, estos no son demasiadas ni muy graves. La deuda técnica era de 6 días, con 293 problemas pero aún así teníamos una calificación A y una tasa de deuda técnica de apenas el 3.7%.

Entre los errores presentados por la aplicación aparecían frecuentemente problema por complejidad ciclomática, código repetido y documentación en el código innecesaria, entre otras. Estos problemas podría causar problemas para alguien que deseara modificar el código pues le tomaría más trabajo entenderlo y corregir los errores en código que estuviese repetido.

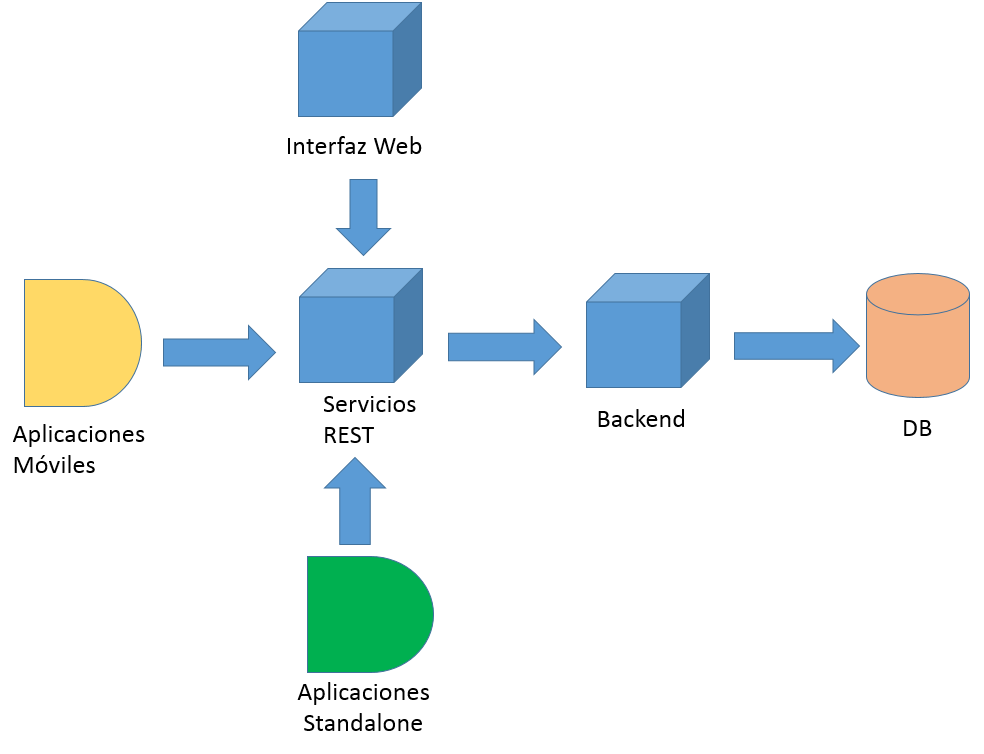
Para reducir las 6 horas de deuda técnica que reportó sonar se aplicaron las técnicas sugeridas por la aplicación. Después de este trabajo se obtuvieron los siguientes resultados:



La complejidad de nuestra aplicación se redujo dramáticamente. La deuda técnica pasó de 6 días a 7 horas, la tasa del 3.7% a solo 0.6% y solo quedaron pendientes 57 problemas. Estos problemas son amenazas menores que no impactan de manera directa la modificabilidad de nuestra aplicación.

Arquitectura JAX-RS

Es una API que ofrece la interfaz de comunicación para facilitar el uso de servicios REST. Su uso garantiza modificabilidad y escalabilidad, ya que usa etiquetas de java SE5 para la simplificación de puntos terminales de comunicación con las aplicaciones consumidoras de servicios. El uso de etiquetas nos permite un mayor desacoplamiento, lo cual nos brindara una reducción de costos en el mantenimiento del código. La estructura que utiliza la aplicación se muestra a continuación:



Por otra parte, el uso de interfaces en cada uno de los componentes favorece una separación de las responsabilidades de cada una de las clases y de la implementación. Esto contribuye a reducir el número de errores al momento de modificar alguno de los componentes.

Documentación

Todos los métodos tanto en la capa de backend como en la capa de servicios se encuentran debidamente documentadas bajo un estándar que garantiza el entendimiento de que hacen los distintos métodos dando una descripción general de estos y el tipo de retorno que este genera. Adicionalmente, incluye la descripción de los parámetros necesarios para la ejecución del mismo y las posibles excepciones que este puede generar con el fin de no solo saber que hace, si no también poder usarlo.

Ésta debida documentación garantiza que a la hora de querer agregar una nueva funcionalidad en TBC sea de fácil entendimiento los diferentes métodos existentes, pudiendo así utilizar los métodos ya creados para implementar la nueva funcionalidad, evitando redundancia de código y minimizando al máximo el tiempo dedicado a entender que se está haciendo en cierta parte del código.

Unidad de persistencia JPA

Este es otro de los puntos favorables respecto a modificabilidad, ya que permite persistir el modelo del mundo una base de datos relacional desacoplando el manejo de la misma. Además de permitir el manejo de metadatos y establecimiento de relaciones entre las entidades. Lo cual brinda mucha potencia en el desarrollo. Ya que extrae al programador de la implementación de comunicación de la misma. Los “entities” son renderizados por el contenedor de aplicaciones y este se encarga del manejo de su ciclos de vida. Lo cual facilita el desarrollo. Y al presentar un código más limpio, favorece la modificabilidad y legibilidad.

Con la ayuda de JPA logramos cambiar, de manera muy sencilla, de la base de datos derby que teníamos a una base de datos Oracle, que es superior en rendimiento y robustez.

Trade-Offs

Debido a que como requerimiento de modificabilidad SonarCube requería la generación de loggers, esto generará un un trade-off con el atributos de calidad de desempeño.

Este atributo se ve afectado debido a que al requerirse una unificación de los loggers de las diferentes máquinas en las que se ejecuta la aplicación, necesariamente se generará un overhead por estas conexiones y la generación de los mismos loggers.

Conclusiones

La modificabilidad de nuestra aplicación se vio dramáticamente mejorada con la ayuda de la herramienta *“sonar cube”,* que a través de una serie de métricas nos ayudó a reducir la complejidad y duplicidad de nuestro código.

Con la arquitectura JAX que poseíamos podemos garantizar que nuestra aplicación posee una alta cohesión entres sus funciones, pues todas los servicios similares se encuentran en las mismas secciones de código, reduciendo el impacto de cambios sobre el proyecto. La arquitectura también favorece el desacoplamiento de las funciones pues poseemos en 3 diferentes proyectos la interfaz de usuario, la exposición de los servicios u el backend.

La documentación también facilitará la modificación del código y la implementación de nuevas funciones en futuros trabajos por realizar sobre el código.

También fue muy útil la implementación de la unidad de persistencia, pues con ella realizamos cambios de la base de datos sin impactar el código que realizamos.

Con las modificaciones hechas en esta etapa la deuda técnica pasó de 6 días a 7 horas, la tasa del 3.7% a solo 0.6% y solo quedaron pendientes 57 problemas. Los cuales según un análisis de costo beneficio no impactan de manera directa la modificabilidad de nuestra aplicación.