Facultad de ingeniería

UNIVERSIDAD NACIONAL DE COLOMBIA

Entrega 06

Documentación Linter Usado

INGENIERÍA DE SOFTWARE 1

Maria Catalina Rodriguez Cardona

Julian David Velandia Neuta

Julian David Albarracin Galindo

Daniel Estiven Blanco Diaz

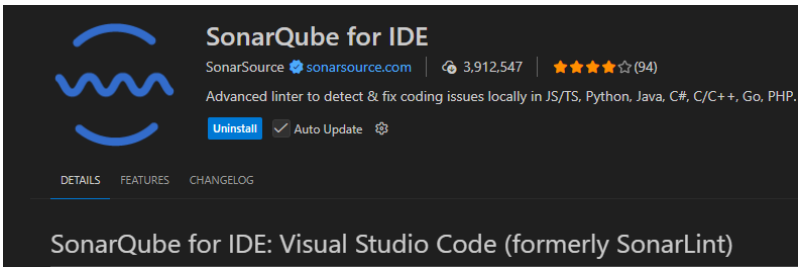
Diego Alejandro Arevalo Guevara

Profesor:

Oscar Eduardo Alvarez Rodriguez

Noviembre 16 de 2025

¿Qué herramienta usaremos?



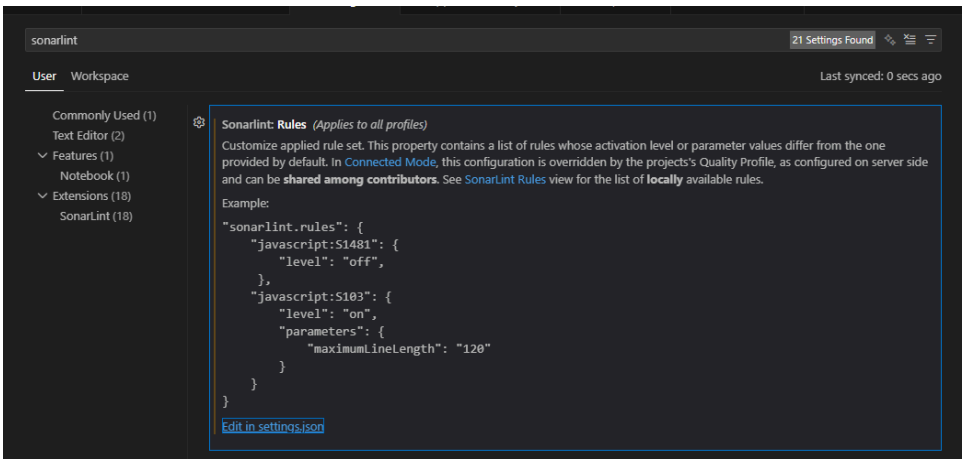
Usamos la herramienta SonarQuebe for IDE, llamada también SonarLint, este es un analizador estático que se descarga como una extensión en el IDE que se utiliza para desarrollar el proyecto, este funciona en múltiples lenguajes, en este caso o estamos usando en Java.

Característica especial, a diferencia de otros linters que se ejecutan con un comando sobre el proyecto para que genere reportes, este se ejecuta al abrir la carpeta raíz del proyecto desde el IDE, ahí pasa a analizar los archivos y así mismo, procede a indicar donde hay errores a la hora de programar y así mismo indica cosas que podrían optimizarse. Más adelante hablaremos de cómo es que este muestra esta información dado que no genera reportes en docs xml como otros linters.

¿Qué configuración tiene?

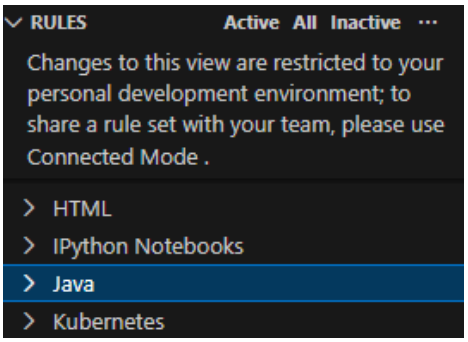
Para el tema de configuración vamos a mostrar como se ve en Visual Studio Code.

Para ver la configuración actual accedemos a la configuración del IDE y miramos lo que diga de SonarLint en la parte de extensiones. En teoria aca deberia mostrar las configuraciones personalizadas que agregue el usuario distintas a las que trae por defecto, en este caso no agregamos ninguna:

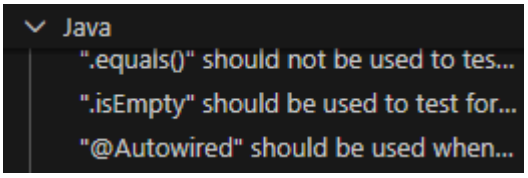


Así mismo es poder todas las configuraciones con las que viene para java dando click en:

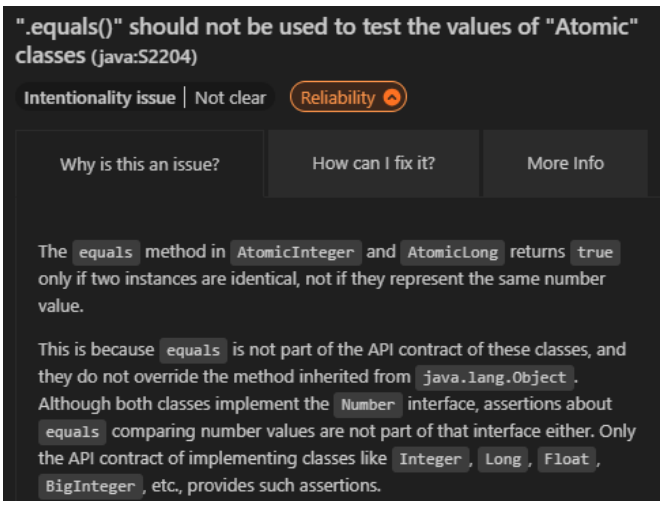
luego escogemos Java:



Y podemos ver todas las disponibles que se están corriendo en el repositorio:



Si le damos click a alguna podemos ver más información:

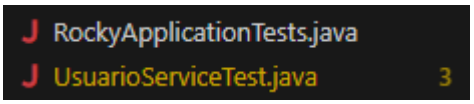


Así mismo existe esta pagina que habla más a profundidad de esto:

https://rules.sonarsource.com/java/

¿Qué resultados obtuvimos con ella?

Nos pinta de amarillito cuando detecta posibles errores (no graves que afectan ejecución) dentro del código, en el siguiente ejemplo pinta de amarillo el nombre de UsuarioServiceTest.java y tambien nos indica de 3 errores.



Así mismo, cuando accedemos al archivo, en la barra de desplazamiento indica en que parte del código esta el error:

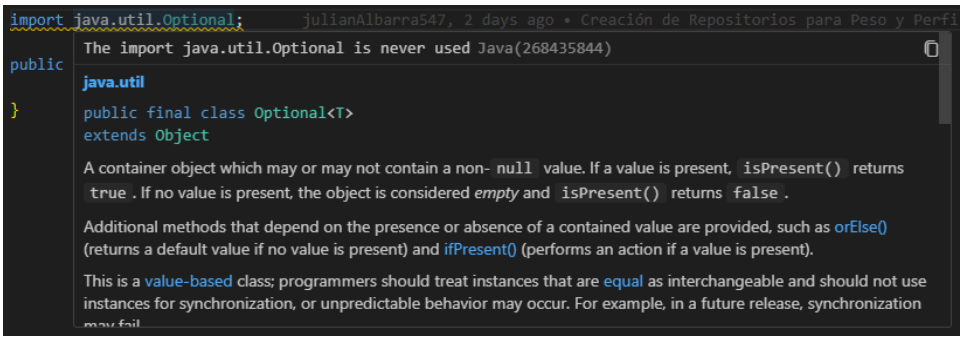
Por ultimo subraya de amarillo la linea de codigo que tiene el error, en este caso (any(Usuario.class)):

```
when(passwordEncoder.encode(u.getPassword())).thenReturn(value: "encoded");
when(repo.save(any(type: Usuario.class))).thenAnswer(invocation -> {
    Usuario guardado = invocation.getArgument(index: 0);
    guardado.setId(id: 1L);
    return guardado;
});
```

¿Cómo se ve en ejecución?

Como vemos en el caso anterior, nos ayuda a detectar errores mientras desarrollamos el código, es decir, antes de ejecutar este, para así ahorrarnos tiempo debugueando y también a corregir posibles errores futuros o relacionados a errores de escritura.

Si nos paramos sobre la línea que señala nos indica el error que posee esa línea, por ejemplo en el siguiente caso, nos indica que la dependencia a pesar de que la cargamos, nunca la usamos, esto ayuda a optimizar recursos dado que así no cargamos cosas inútiles para el proyecto, es decir, que no hacemos uso de estas:



¿Cómo nos ayuda a corregir errores?

Ayuda a corregir errores que se cometen durante la etapa de desarrollo, optimizando tiempos dado que no tenemos que levantar contenedores, cargar dependencias otras cosas para ver si funciona o no el código, aun así, a pesar de ser tan buena, esta no es perfecta y requiere ser usada con precaución