



REPORTE LINT



E2.01

Resumen	3
Tabla de contenidos	3
Contenidos	3
Conclusiones	5
Bibliografía	5

Resumen

En este documento se detalla el análisis del código fuente del proyecto a través del plugin SonarLint usado en la asignatura. Sonar nos permite la detección de posibles bugs o code smells(código que en un futuro puede ocasionar problemas si no son subsanados) que se han producido durante el desarrollo. Precisamente, en este documento se listan los bugs y code smells que no se eliminan del sistema y su razón.

Tabla de contenidos

Versión: 1.0	22/05/2022	Creación del documento
--------------	------------	------------------------

Contenidos

En cuanto a los bugs y code smells detectados:

```
@Override
public void validate(final Request<SystemConfiguration> request, final SystemConfiguration entity, final Errors errors) {
    assert request != null;
    assert entity != null;
    assert errors != null;
    String[] currencies;
    int k=0;
    if(!errors.hasErrors("defaultSystemCurrency")) {
        if(entity.getAcceptedCurrencies()!=null) {
            currencies = entity.getAcceptedCurrencies().split(",");
            for(final String currency:currencies) {
                if(entity.getDefaultSystemCurrency().equals(currency)) ++k;
            }
            errors.state(request, k>0, "defaultSystemCurrency", "administrator.system-configuration.form.error.not-default-currency");
        }
    }
}
```

Figura 1.

El primer y único code smell detectado se encontraba en la clase “AdministratorSystemConfigurationPerformService” y destaca que el uso de cadenas de condicionales no es recomendable y puede ser grave si esta cadena es más grande, lo que dificultaría la legibilidad del código.

Para solucionar este code smell sería necesario agrupar los dos condicionales en uno y crear un solo booleano con la operación And “&&”. En este caso sería factible realizarlo debido a que la cantidad de condicionales es escasa.

```

@Override
public void validate(final Request<Item> request, final Item entity, final Errors errors) {
    assert request != null;
    assert entity != null;
    assert errors != null;
    if(!errors.hasErrors("code")) {
        Item item;
        item = this.repository.findItemByCode(entity.getCode());
        errors.state(request, item == null || item.getCode() == entity.getCode(), "code", "inventor.item.form.error.duplicated");

        boolean descriptionWithinThreshold, nameWithinThreshold, technologyWithinThreshold;

        descriptionWithinThreshold = SpamCheck.isWithinSpamThreshold(entity.getDescription());
        nameWithinThreshold = SpamCheck.isWithinSpamThreshold(entity.getName());
        technologyWithinThreshold = SpamCheck.isWithinSpamThreshold(entity.getTechnology());

        errors.state(request, !descriptionWithinThreshold, "description", "inventor.item.form.error.spam");
        errors.state(request, !nameWithinThreshold, "name", "inventor.item.form.error.spam");
        errors.state(request, !technologyWithinThreshold, "technology", "inventor.item.form.error.spam");

    }
    if (!errors.hasErrors("retailPrice")) {
        final Money budget = entity.getRetailPrice();
        final Boolean isBudgetOverZero = budget.getAmount() > 0.;
        final String[] splits = this.repository.findAcceptedCurrencies().split(",");
        Boolean isCurrencyAccepted;
        isCurrencyAccepted = false;
        for (int i = 0; i < splits.length; i++) {
            if (splits[i].equals(budget.getCurrency())) {
                isCurrencyAccepted = true;
            }
        }
    }
}

```

Figura 2.

```

@Override
public void validate(final Request<ToolKit> request, final ToolKit entity, final Errors errors) {
    assert request != null;
    assert entity != null;
    assert errors != null;
    if(!errors.hasErrors("code")) {
        ToolKit toolkit;
        toolkit = this.repository.findToolKitByCode(entity.getCode());
        errors.state(request, toolkit == null || toolkit.getCode() == entity.getCode(), "code", "inventor.tool-kit.form.error.duplicated");

        boolean descriptionWithinThreshold, titleWithinThreshold, assemblyNotesWithinThreshold;

        descriptionWithinThreshold = SpamCheck.isWithinSpamThreshold(entity.getDescription());
        titleWithinThreshold = SpamCheck.isWithinSpamThreshold(entity.getTitle());
        assemblyNotesWithinThreshold = SpamCheck.isWithinSpamThreshold(entity.getAssemblyNotes());

        errors.state(request, !descriptionWithinThreshold, "description", "inventor.tool-kit.form.error.spam");
        errors.state(request, !titleWithinThreshold, "name", "inventor.tool-kit.form.error.spam");
        errors.state(request, !assemblyNotesWithinThreshold, "assemblyNotes", "inventor.tool-kit.form.error.spam");

    }
}

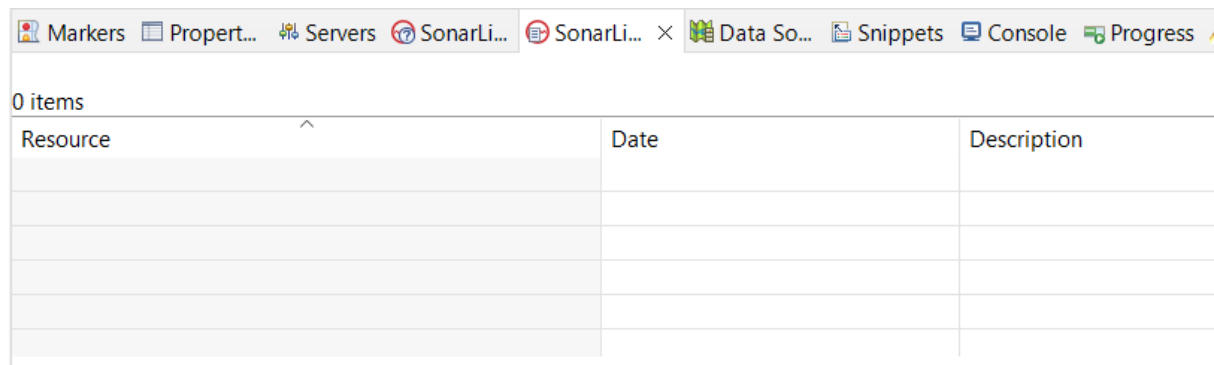
```

Figura 3.

Los bugs detectados en las figuras 2 y 3 se localizaban en las clases “InventorItemCreateService” y “InventorToolKitCreateService” y denotan el uso de comparaciones con el operando “==” debido a que no se realiza una comparativa de los valores, sino de las localizaciones en la memoria.

En cuanto a la resolución de este tipo de bug es necesario compararlos con el método equals para que realmente compare los valores de las cadenas.

Tras la refactorización aplicada, los bugs y code smells detectados fueron eliminados, tal y como se muestra en la siguiente figura:



Resource	Date	Description

Figura 4.

Conclusiones

No se han detectado una gran cantidad de bugs y code smells, lo que ha facilitado la aplicación de una refactorización para solucionarlos.

Uno de los factores que ha ocasionado que no se hayan producido una gran cantidad de estos es debido al uso del plugin de SonarLint en el entorno de desarrollo Eclipse, lo que ha producido que si se ha creado alguno, este sea solventado antes de su subida a producción.

Bibliografía

No aplica