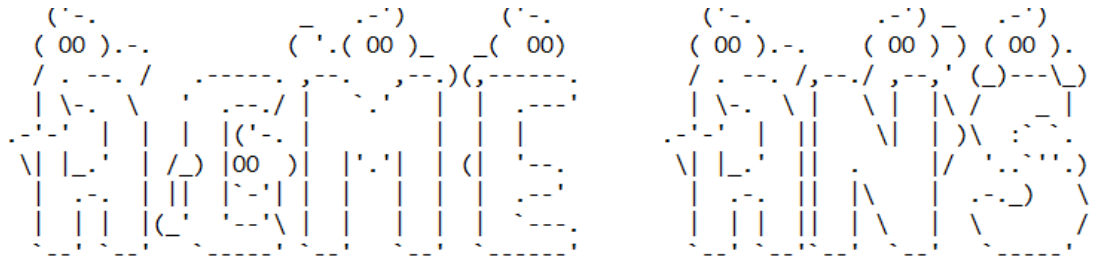
REPORTE DE LINT

Acme-ANS-D04



Repositorio: https://github.com/FranciscoFernandezN/Acme-ANS

Creado por el grupo C1.022, del G1

Participantes	
Nombres	Correos
Gutiérrez Arazo, Beatriz	beagutara@alum.us.es

25 de mayo de 2025

Índice

Resumen ejecutivo

Este es el proyecto del grupo C1.022 sobre Acme AirNav Solutions, S.A. (Acme ANS, S.A. abreviado), la cual es una compañía ficticia especializada en ayudar a aeropuertos a organizar y coordinar sus operaciones a partir de soluciones desarrolladas en software. La logística de los vuelos (la programación de los vuelos, la organización de reservas y de tripulación, etc.) se gestionan mediante el desarrollo de un WIS.

Con esto, para el correcto funcionamiento de la aplicación, se deberá escribir un reporte de testing donde se describan los valores de cobertura del código junto con los valores correspondientes al rendimiento de la aplicación, ambos, previos y posteriores a realizar mutaciones en el código.

Tabla de revisiones

Número	Fecha	Descripción
v1.0.0	25/05/2025	Versión finalizada del documento para el entregable 4

Introducción

Para el contenido de este documento, se han tenido en cuenta todos los resultados de SonarLint para los requisitos individuales del estudiante 2, tanto de aquellos requisitos obligatorios como opcionales.

Para cada entrada relevante de SonarLint, explicaré por qué pienso que no debe ser cambiado, o en su defecto, por qué lo he cambiado.

Contenidos

Resource	Date	Description
CustomerBookingController.java	1 month ago	Remove this field injection and use constructor injection instead.
CustomerBookingController.java	1 month ago	Remove this field injection and use constructor injection instead.
CustomerBookingController.java	1 month ago	Remove this field injection and use constructor injection instead.
CustomerBookingCreateService.java	12 days ago	Define a constant instead of duplicating this literal "passenger" 9 times. [+9 loca...
CustomerBookingCreateService.java	19 days ago	Define a constant instead of duplicating this literal "locatorCode" 6 times. [+6 lo...
CustomerBookingCreateService.java	1 month ago	Use a StringBuilder instead.
CustomerBookingCreateService.java	1 month ago	Define a constant instead of duplicating this literal "travelClass" 3 times. [+3 loca...
CustomerBookingCreateService.java	1 month ago	Remove this field injection and use constructor injection instead.
CustomerBookingCreateService.java	1 month ago	Define a constant instead of duplicating this literal "flight" 8 times. [+8 locations]
CustomerBookingListService.java	2 days ago	Use a primitive boolean expression here.
CustomerBookingListService.java	1 month ago	Remove this field injection and use constructor injection instead.
CustomerBookingPublishService.java	29 days ago	Define a constant instead of duplicating this literal "flight" 8 times. [+8 locations]
CustomerBookingPublishService.java	29 days ago	Define a constant instead of duplicating this literal "passenger" 11 times. [+11 lo...
CustomerBookingPublishService.java	29 days ago	Define a constant instead of duplicating this literal "travelClass" 3 times. [+3 loca...
CustomerBookingPublishService.java	1 month ago	Use a StringBuilder instead.
CustomerBookingPublishService.java	1 month ago	Remove this field injection and use constructor injection instead.
CustomerBookingPublishService.java	1 month ago	Define a constant instead of duplicating this literal "lastNibble" 4 times. [+4 loca...
CustomerBookingPublishService.java	1 month ago	Define a constant instead of duplicating this literal "locatorCode" 6 times. [+6 lo...
CustomerBookingPublishService.java	1 month ago	Refactor this method to reduce its Cognitive Complexity from 16 to the 15 allow...
CustomerBookingShowService.java	1 month ago	Remove this field injection and use constructor injection instead.
CustomerBookingUpdateService.java	12 days ago	Define a constant instead of duplicating this literal "passenger" 9 times. [+9 loca...
CustomerBookingUpdateService.java	19 days ago	Define a constant instead of duplicating this literal "locatorCode" 4 times. [+4 lo...
CustomerBookingUpdateService.java	29 days ago	Define a constant instead of duplicating this literal "travelClass" 3 times. [+3 loca...
CustomerBookingUpdateService.java	1 month ago	Remove this field injection and use constructor injection instead.
CustomerBookingUpdateService.java	1 month ago	Define a constant instead of duplicating this literal "flight" 5 times. [+5 locations]
CustomerPassengerController.java	1 month ago	Reorder the modifiers to comply with the Java Language Specification.
CustomerPassengerController.java	1 month ago	Remove this field injection and use constructor injection instead.
CustomerPassengerController.java	1 month ago	Remove this field injection and use constructor injection instead.
CustomerPassengerController.java	1 month ago	Remove this field injection and use constructor injection instead.
CustomerPassengerController.java	1 month ago	Remove this field injection and use constructor injection instead.
CustomerPassengerController.java	1 month ago	Remove this field injection and use constructor injection instead.
CustomerPassengerCreateService.java	19 days ago	Define a constant instead of duplicating this literal "booking" 11 times. [+11 loca...
CustomerPassengerCreateService.java	1 month ago	Define a constant instead of duplicating this literal "passportNumber" 3 times. [+...
CustomerPassengerCreateService.java	1 month ago	Remove this field injection and use constructor injection instead.
CustomerPassengerListService.java	2 days ago	Use a primitive boolean expression here.
CustomerPassengerListService.java	1 month ago	Reorder the modifiers to comply with the Java Language Specification.
CustomerPassengerListService.java	1 month ago	Remove this field injection and use constructor injection instead.
CustomerPassengerPublishService.java	1 month ago	Remove this field injection and use constructor injection instead.
CustomerPassengerPublishService.java	1 month ago	Define a constant instead of duplicating this literal "booking" 12 times. [+12 loca...
CustomerPassengerPublishService.java	1 month ago	Define a constant instead of duplicating this literal "passportNumber" 4 times. [+...
CustomerPassengerShowService.java	1 month ago	Remove this field injection and use constructor injection instead.
CustomerPassengerUpdateService.java	10 days ago	Define a constant instead of duplicating this literal "booking" 10 times. [+10 loca...
CustomerPassengerUpdateService.java	1 month ago	Define a constant instead of duplicating this literal "passportNumber" 3 times. [+...
CustomerPassengerUpdateService.java	1 month ago	Remove this field injection and use constructor injection instead.
CustomerRecommendationController.java	1 day ago	Remove this field injection and use constructor injection instead.
CustomerRecommendationController.java	16 days ago	Remove this field injection and use constructor injection instead.
CustomerRecommendationController.java	16 days ago	Remove this field injection and use constructor injection instead.
CustomerRecommendationListRelatedService.java	1 day ago	Use a primitive boolean expression here.
CustomerRecommendationListRelatedService.java	1 day ago	Remove this field injection and use constructor injection instead.
CustomerRecommendationListService.java	2 days ago	Use a primitive boolean expression here.
CustomerRecommendationListService.java	16 days ago	Remove this field injection and use constructor injection instead.
CustomerRecommendationShowService.java	16 days ago	Remove this field injection and use constructor injection instead.

Figura 1 y 2: Resultados SonarLint para las features de Customer

Antes de empezar a hablar de los bad smells encontrados, quiero mencionar que NO he realizado ningún cambio en base a SonarLint. A continuación, explicaré los motivos.

```java
@GuiController
public class CustomerBookingController extends AbstractGuiController<Customer, Booking> {

    // Internal state ---------------------------------------------------------

    @Autowired
    private CustomerBookingListService      listService;

    @Autowired
    private CustomerBookingShowService      showService;

    @Autowired
    private CustomerBookingCreateService    createService;

    @Autowired
    private CustomerBookingUpdateService    updateService;

    @Autowired
    private CustomerBookingPublishService   publishService;
```

Figura 3: @Autowired

Por una parte, un bad smell recurrente es el uso del @Autowired directamente en atributos en vez de realizarlo sobre un constructor. Si bien esto podría ser un problema, hemos estado realizando en base al ejemplo Acme-Jobs proporcionado por los profesores, y no considero conveniente tratar de hacer cosas distintas a como se nos ha enseñado en la asignatura.

```
        SelectChoices travelClasses;
        Collection<Flight> flights = this.repository.findAllFlightsForBooking();

        travelClasses = SelectChoices.from(TravelClass.class, booking.getTravelClass());

        SelectChoices flightChoices = new SelectChoices();
        int flightId = super.getRequest().hasData("flight") ? super.getRequest().getData("flight", int.class) : -1;
        flights.stream().forEach(f -> flightChoices.add(String.valueOf(f.getId()),
            String.format("%s - %s, %s - %s, %s, %s", f.getOrigin(), f.getDestiny(), f.getScheduledDeparture(), f.getScheduledArriva
        flightChoices.add("0", "----", flightId <= 0);

        Collection<Passenger> passengers = this.repository.findPassengersByCustomerId(customerId);
        passengers.removeAll(this.repository.findPassengersByBookingId(booking.getId()));
        SelectChoices passengerChoices = new SelectChoices();
        Duplication | Duplication
        int passengerId = super.getRequest().hasData( 7 "passenger") ? super.getRequest().getData( 8 "passenger", int.class) : -1;
        passengers.stream().distinct().forEach(p -> passengerChoices.add(String.valueOf((Integer) p.getId()), String.format("%s - %s
        passengerChoices.add("0", "----", passengerId <= 0);

        dataset = super.unbindObject(booking, "travelClass", "lastNibble", "isDraftMode");
        dataset.put("locatorCode", locatorCode);
        dataset.put("travelClasses", travelClasses);
        dataset.put("flight", "0");
        dataset.put("flightChoices", flightChoices);
        Duplication
        dataset.put( 9 "passenger", passengerId);
        dataset.put("passengerChoices", passengerChoices);
        dataset.put("updatedBooking", false);
        dataset.put("purchaseMoment", MomentHelper.getCurrentMoment());

        super.getResponse().addData(dataset);
    }
```

Figura 4: repetición de literales

Por otra parte, también ocurre bastante la repetición de literales (como su nombre indica, significa escribir varias veces un mismo literal en una clase en vez de declararla como variable global). Esto supondría un gran problema si decidiera, en este caso, sustituir “passenger” por “passengerId” porque pensara que es intuitivo: tendría que cambiar “passenger” de los 9 sitios, y si se me olvidara en alguno ocurriría errores. Decidí no cambiarlo debido a que escogí bien los nombres inicialmente y no he tenido la necesidad de cambiarlo.

Por último, el resto de bad smells no han sido consideradas de tanta importancia como estos. El motivo por el que no he “corregido” nada es que, desde que empecé el proyecto, he tenido la herramienta de SonarLint instalada (como venía explicada en la guía). Por tanto, a medida que he ido programando, cualquier bad smell importante me lo ha notificado la herramienta con su subrayado azul y lo he podido corregir. Es decir, soy consciente de todos los bad smells que se encuentran ahora en el proyecto por mis features, y de que no tendrán repercusiones en este (o, en todo caso, serán leves).

Conclusiones

Como conclusión, el reporte Lint es muy útil para poder visualizar bad smells en el código que puedan suponer problemas de mantenimiento o introducción de bugs en el futuro. Si bien aquí no le di demasiado uso al reporte como tal, sí considero que tiene mucha importancia (tanto la herramienta como el reporte) a la hora de llevar a cabo un proyecto software, pues la presencia de bad smells es determinante para un desarrollo en el que hacer cambios no suponga una rotura de la aplicación.

Bibliografía

Intencionalmente en blanco