LINT REPORT

Raquel García Hortal - Student 2

Table of Contents

1. EXECUTIVE SUMMARY

This report summarizes the static code analysis results provided by SonarLint for our project. The tool has detected several minor issues known as code smells, which may potentially affect maintainability, readability, or performance. Upon careful review, we identified that most of the reported issues are either false positives or harmless in the context of our framework. This document presents the analysis and our justifications.

2. REVISION TABLE

Revision Number	Date	Description
1.0.0	01/07/2025	Create the lint report

3. INTRODUCTION

Linting tools help developers identify potential problems in the source code before they become bugs or create maintenance issues. SonarLint, in particular, integrates directly into the IDE and provides real-time feedback on code quality. In this report, we review the alerts detected by SonarLint in our project, focusing on those that were deemed non-critical or innocuous. Any warnings raised by the Acme Framework have been safely ignored, as instructed.

4. IDENTIFIED BAD SMELLS

4.1 Remove this useless assignment to local variable 'bookings'

File: AdministratorBookingListService.java and CustomerBookingListService.java

Line: Collection<Booking> bookings = new ArrayList<>();

Severity: Minor (warning - yellow)

Sonar Message: "Remove this useless assignment to local variable 'bookings'."

Description: SonarLint suggests removing the initial assignment of bookings = new ArrayList<>() because the variable is immediately reassigned with data from the repository.

Justification: This assignment is innocuous. While technically redundant, it does not introduce any side effect or performance issue. It can help with readability and is useful for anticipating future changes or debugging. Therefore, we have decided to keep this line for clarity and maintainability reasons.

4.2 Define a constant instead of duplicating this literal 'bookingId' 4 times

File: CustomerPassengerCreateService.java

Occurrences: 4

Severity: Major (red)

Sonar Message: "Define a constant instead of duplicating this literal 'bookingId' 4 times."

Description: SonarLint detects that the string literal "bookingId" is repeated four times in the file, and recommends replacing it with a class-level constant to follow the DRY (Don't Repeat Yourself) principle.

Justification: This literal appears only a few times and always in closely related and localized code blocks. Introducing a constant would add minimal value in this case and slightly reduce code readability. The context and usage make the repetition harmless and easily understandable. Therefore, we consider this duplication innocuous and have chosen to keep the current form for simplicity.

4.3 Remove this field injection and use constructor injection instead

File: CustomerPassengerUpdateService.java

Location: Field @Autowired private CustomerPassengerRepository repository;

Severity: Minor (warning - yellow)

Sonar Message: "Remove this field injection and use constructor injection instead."

Description: SonarLint suggests replacing field injection using @Autowired with constructor injection to improve testability, immutability, and make dependencies more explicit.

Justification: Although constructor injection is generally preferred in modern Spring-based applications, we consider field injection acceptable in this context because:

- The class is already tightly coupled with the framework and not meant to be tested in isolation.
- The service class is short-lived and has only one dependency.
- All other services in the project follow the same field-injection pattern, maintaining consistency across the codebase.

Therefore, replacing it with constructor injection would add unnecessary boilerplate without practical benefit in this case.

4.4 Override the equals method in this class.

Files: Booking.java, Passenger.java, BookingRecord.java and Customer.java

Severity: Info (blue)

Sonar Message: "Override the equals method in this class."

Description: SonarLint suggests overriding the equals method in entity classes to ensure proper object comparison, especially when such objects are stored in collections or compared in business logic.

Justification:

In this project:

- These classes are managed entirely by JPA and the ACME Framework, which uses entity IDs and persistence context for comparison and uniqueness.
- Booking, Passenger and BookingRecord extend AbstractEntity, and Customer extends AbstractRole, both of which already provide proper equals() and hashCode() implementations based on the entity's identity.

Adding a custom equals() implementation would be redundant and potentially error-prone, as the framework already handles this through its inheritance hierarchy.

Hence, this SonarLint issue is deemed innocuous and intentionally ignored.

4.5 Use isEmpty() to check whether the collection is empty or not

File: CustomerBookingPublishService.java

Severity: Info (blue)

Sonar Message: "Use isEmpty() to check whether the collection is empty or not."

Description: The code uses passengers.size() > 0 instead of !passengers.isEmpty().

Justification: While using isEmpty() is preferred for clarity, the current usage is explicit and consistent with the codebase style. This has no performance or functional impact and is considered an innocuous bad smell.

5. CONCLUSIONS

After evaluating all bad smells reported by SonarLint, the team determined that the issues listed above do not compromise the functionality or maintainability of the project. Most of them are either stylistic or constrained by framework conventions. As such, no code modifications were deemed necessary.

6. BIBLIOGRAPHY

Intentionally blank.