# **Testing Report**



|  |
| --- |
| **Group:** C1.031 |
| **Repository:** https://github.com/aaronma300604/DP2-C01-31 |
| **Members:**  **Student #1:** López Ramos, Daniel ([danlopram@alum.us.es](mailto:danlopram@alum.us.es))  **Student #2:** Ángel Postigo, Estrella ([estangpos@alum.us.es](mailto:estangpos@alum.us.es))  **Student #3:** Miranda Balastegui, Iván ([ivamirbal@alum.us.es](mailto:ivamirbal@alum.us.es))  **Student #4:** Terrón Hernández, Diego ([dieterher@alum.us.es](mailto:dieterher@alum.us.es))  **Student #5:** Mayoral Ansias, Aarón ([aarmayans@alum.us.es](mailto:aarmayans@alum.us.es)) |
| **Date:** Seville May 20, 2025 |

Table of Contents

[**Testing Report** 1](#_Toc199081253)

[**Revision Table** 3](#_Toc199081254)

[**Introduction** 3](#_Toc199081255)

[**Contents** 3](#_Toc199081256)

[**Functional Testing** 3](#_Toc199081257)

[**Performance Testing** 5](#_Toc199081258)

[**Conclusion** 10](#_Toc199081259)

[**Bibliography** 10](#_Toc199081260)

# **Revision Table**

|  |  |  |
| --- | --- | --- |
| **Revision Number** | **Date** | **Description** |
| 1.0 | 25/05/2025 | Intial version – all sections added |

# **Introduction**

The purpose of this document is to provide an analysis of the various tests that have been carried out to assess code coverage related to Student 2, as well as an analysis of the system's performance with respect to the requirements associated with that student.

# **Contents**

## **Functional Testing**

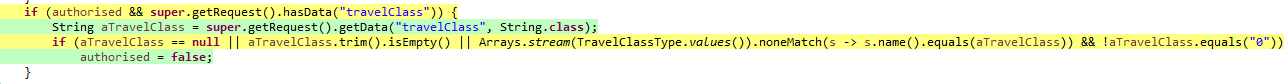
The following is a detailed list of the functional tests performed on the **Booking** entity:

* **List.safe:** This test verifies that a Customer can view all of their Booking records.
* **Show.safe:** This test ensures that an individual Booking can be viewed. If the Booking is in draft mode, it can be edited, updated, or deleted. Once it is published, it becomes immutable—neither deletable nor updatable. A Customer can only view their own Booking records, whether published or not.
* **Create.safe:** Several Booking records are created using valid data accepted by the system. Additionally, an attempt is made to create a Booking without providing any input, triggering the appropriate error messages. Finally, valid data is combined with invalid values to test constraints on Date fields (out-of-range values) and String fields (excessively long messages). A Booking must always be associated with a flight that is already published.
* **Update.safe:** Multiple Booking records belonging to the Customer are updated using both valid and invalid data, following a process similar to Create.safe.
* **Delete.safe**: A Booking can be deleted as long as it has not been published. If a Booking is deleted, the intermediate entity linking it to a Passenger must also be removed.
* **Publish.safe:** A Booking can only be published if all field values are valid and it has at least one associated Passenger, which must also be published.
* **Ids.hack**: Attempts to manipulate IDs have been prevented, both via the URL and through browser inspection tools. Invalid values such as 0 or 999 are not allowed in operations such as Show, Update, Delete, and Publish. For Create, the ID must be exactly 0. It has also been verified that a Customer cannot access Booking records belonging to others, regardless of publication status. Additionally, malformed URLs (e.g., changing booking/show?id=33 to booking/show) are correctly handled.
* **Show.hack**: GET request hacking tests were performed to attempt access to Booking records belonging to other Customers, both published and in draft mode.
* **Create-update.hack:** POST request hacking tests were conducted to simulate the creation or update of a Booking associated with a non-existent flight (e.g., ID '999') or with an unpublished flight, using developer tools such as F12. The system consistently returned the expected error responses.
* **Delete.hack:** POST request hacking tests were executed to attempt deletion of Booking records belonging to other Customers, both published and unpublished. It was also verified that a published Booking cannot be deleted.

The test coverage achieved for the Booking entity is **99.4%.**

Gráfico

El contenido generado por IA puede ser incorrecto.

All lines of this entity have been evaluated either fully or partially. Full coverage is not achieved primarily due to a line in the authorise method, which includes an OR condition that is not fully exercised. This line prevents POST hacking attempts involving invalid enumeration values. Since this validation is repeated across Create, Delete, and Publish methods, it slightly lowers the overall test coverage.

The following is a detailed list of the functional tests performed on the **Passenger entity:**

* **List.safe:** This test verifies that a Customer can view all their Passenger records.
* **Show.safe:** This test ensures that an individual Passenger can be viewed. If it is in draft mode, it can be edited, updated, or deleted. Once published, it can no longer be modified or removed. A Customer can only view their own Passenger records, whether published or not.
* **Create.safe**: Several Passenger records are created using valid data accepted by the system. An attempt is made to create a Passenger without providing any input, resulting in the corresponding error messages. Then, valid data is combined with invalid values to test constraints on Date fields (out-of-range values) and String fields (overly long messages).
* **Update.safe:** Multiple Passenger records belonging to the Customer are updated using both valid and invalid inputs, similarly to the Create.safe test.
* **Delete.safe**: A Passenger may be deleted as long as it has not been published. When a Passenger is deleted, the intermediate entity linking it to a Booking must also be removed.
* **Publish.safe:** A Passenger can only be published if all field values are valid.
* **Ids.hack:** ID manipulation has been prevented both via the URL and through browser inspection tools. Invalid values such as 0 or 999 are not permitted for the Show, Update, Delete, and Publish operations. For Create, the ID must be exactly 0. It has also been verified that a Customer cannot access Passenger records belonging to others, regardless of their publication status. Additionally, access has been prevented via incomplete or malformed URLs, such as changing passenger/show?id=33 to passenger/show.
* **Show.hack:** GET request hacking tests were performed to attempt access to Passenger records belonging to other Customers, both published and in draft mode.
* **Create-update.hack:** POST request hacking tests were conducted to attempt creation or modification of Passenger records belonging to other Customers. It was also verified that a published Passenger cannot be updated.
* **Interfaz de usuario gráfica

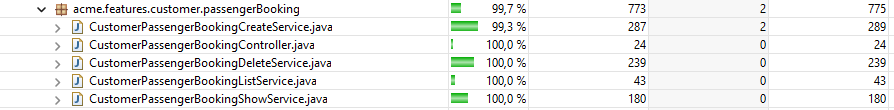
  El contenido generado por IA puede ser incorrecto.Delete.hack:** POST request hacking tests were performed to attempt deletion of Passenger records belonging to other Customers, both published and unpublished. It was also verified that a published Passenger cannot be deleted.

The test coverage achieved for the Passenger entity is **99.0%**.

Texto

El contenido generado por IA puede ser incorrecto.All lines of this entity have been evaluated either fully or partially, except for one line in the Create functionality. This line was not executed due to either insufficient test coverage or a logical error.

The following is a detailed list of the functional tests performed on the **PassengerBooking** entity, which serves as an intermediate relation between Passenger and Booking:

* **List.safe:** This test verifies that a Customer can view all their PassengerBooking relations.
* **Show.safe:** This test checks that a single PassengerBooking relation can be viewed. Depending on whether the associated Booking is published, the delete button may or may not be displayed.
* **Create.safe:** In this intermediate entity, neither the Passenger nor the Booking fields can be null. Additionally, duplicate relations are not allowed.
* **Delete.safe**: A PassengerBooking can be deleted as long as the associated Booking has not been published. If either a Passenger or a Booking is deleted, the intermediate entity linking them must also be removed.
* **Ids.hack:** Attempts to manipulate IDs have been prevented, both via the URL and browser inspection tools. Invalid values such as 0 or 999 are not permitted for Show and Delete operations. For Create, the ID must be exactly 0. It has also been verified that a Customer cannot access PassengerBooking relations belonging to others. Furthermore, malformed URL attempts (e.g., passengerBooking/show?id=33 to passengerBooking/show) are also handled appropriately.
* **Show.hack:** GET request hacking tests were performed to attempt access to PassengerBooking relations belonging to other Customers.
* **Create-update.hack**: POST request hacking tests were performed to prevent the creation of intermediate relations involving an unpublished Passenger and a published Booking. It was also verified that relations cannot be created with Passenger or Booking records owned by other Customers.
* **Delete.hack:** POST request hacking tests were executed to attempt deletion of PassengerBooking relations belonging to other Customers. Additionally, it was verified that a PassengerBooking cannot be deleted if the associated Booking is already published.

The test coverage achieved for the PassengerBooking entity is **99.7%.**

All lines of this entity have been evaluated either fully or partially. However, some lines do not cover all logical conditions, which accounts for the slight reduction from full coverage.Texto

El contenido generado por IA puede ser incorrecto.

## **Performance Testing**

A performance analysis of the system will now be carried out through the execution of the previously mentioned functional tests. The tests have been executed under two different scenarios:

1. **Without query optimization indexes:** The Excel file *“tester-performance-clean-before.xlsx”* contains the average performance results of the test operations as illustrated in the following chart.

As observed, the most time-consuming operation on average is Booking creation, with an execution time of less than 50 milliseconds.

Below are some statistics regarding the **operations**:Tabla

El contenido generado por IA puede ser incorrecto.

The system without indexes presents a confidence interval ranging from 16.14 ms to 13.9 ms.

1. **With query optimization indexes:** The Excel file “*tester-performance-clean-after.xlsx*” contains the average performance results of the test operations, as illustrated in the following chart.

As shown, the most time-consuming operation continues to be **Booking creation**, although the average time has decreased to 35 milliseconds. All operations show improved response times compared to the previous analysis.

Tabla

El contenido generado por IA puede ser incorrecto.Below are some statistics regarding the **operations**:

As we can observe, the system with indexes gives a confidence interval ranging from 10.46 milliseconds to 8.96 milliseconds.

1. Next, using both confidence intervals, a hypothesis test will be conducted using a **z-test**. This test is documented in the file “*analysis-performance.xlsx*”, with supporting screenshots provided below:

Tabla

El contenido generado por IA puede ser incorrecto.

Tabla

El contenido generado por IA puede ser incorrecto.

The p-value obtained from the z-test is 0.1 \* 10-13, indicating that the optimization applied to the system has resulted in a statistically significant improvement in performance.

The average response time decreased by approximately 35%, from 15.01 ms to 9.71 ms. Furthermore, the calculated z-score (7.70) significantly exceeds the critical value, confirming the statistical validity of the improvement.

# **Conclusion**

The functional tests have demonstrated that core operations (listing, viewing, creating, updating, deleting, and publishing) are correctly implemented and properly protected against unauthorized access or external manipulation. Validation mechanisms, both client-side and server-side, responded appropriately to valid and invalid inputs, thereby ensuring the robustness of the system.

Regarding performance, the system exhibited a significant improvement after the implementation of optimization techniques such as the introduction of database indexes. This improvement was confirmed through a statistical z-test, which validated the substantial reduction in response times, especially for more demanding operations such as Booking creation.

# **Bibliography**

Intentionally blank