**INFORME DE ANÁLISIS**



**Grado en Ingeniería Informática – Ingeniería del**

**Software**

**Diseño y Pruebas 2**

**Curso 2024-2025**

Lucía Campos Díez

Índice

Contenido

[1. Executive Summary 4](#_Toc11058)

[2. Review Table 5](#_Toc11059)

[3. Introduction 6](#_Toc11060)

[4. Contents 7](#_Toc11061)

[4.1 Analysis of the individual requirements 7](#_Toc11062)

[5. Conclusions 10](#_Toc11063)

[**6. Bibliography** 11](#_Toc11064)

# Executive Summary

The analysis of the requirements is crucial in the developments of software systems, it makes us capable of measuring and documenting the requirements of the interested parts. This documents shows the final analysis of the student 5 in the last deliverable.

# Review Table

|  |  |  |
| --- | --- | --- |
| Version | Description | Date |
| v1.0 | Initial version | 02/07/2025 |
| v2.0 | Initial version | 02/07/2025 |

# Introduction

The document provides a detailed analysis of the requirements for the fourth individual project deliverable, aiming to make them as clear as possible. First, the requirements will be presented, followed by an explanation of any issues that may have arisen during their resolution, along with the reasoning behind the decisions made.

# Contents

## Analysis of the individual requirements

**MANDATORY REQUIREMENTS**

**D02:**

The **technicians** care of aircraft maintenance by conducting regular inspections, performing repairs, and carrying out other maintenance tasks. The system must store the following data about them: a **license number** (unique, pattern "^[A-Z]{2-3}\d{6}$"), a **phone number** (pattern "^\+?\d{6,15}$"), their **specialisation** (up to 50 characters), whether they have passed their **annual health test** or not, and their **years of experience**. Optionally, the system may store their **certifications** (up to 255 characters).

**Alternative 1:** Implement the Technician model with both attribute-level and class-level validation

While developing the model for this entity, we encountered a particular issue with the validation of the license number field. In our team, we have consistently referred to the license number as codigo, since this terminology has been uniformly used across all similar Realm entities throughout the project.

Initially, it seemed sufficient to apply a regular expression validator at the attribute level. However, during development we realized that the validation of the license number sometimes required access to more context than just the single field — for example, it might involve logic dependent on other parts of the entity or additional constraints from the Realm.

Therefore, we decided that it made much more sense to handle this validation at the **class level**, within the Realm-specific validator, while still applying **attribute-level validation** to enforce the required pattern on codigo.

**Pros:**

* Allows context-aware validation involving the full state of the class, not just the individual field.
* Maintains consistency across all realm implementations in the project.
* Combines pattern matching and logical consistency checks.

**Cons:**

* Intentionally left blank.

After the analysis, **Alternative 1 was chosen as the solution** for implementing the Technician entity model.

**D03:**

8)Operations by **technicians** on **maintenance records**:

* List their maintenance records.
* Show the details of the maintenance records, including the tasks involved.
* Create, update, and publish maintenance records. Please, note that to publish a maintenance record, it cannot have any unpublished tasks and should have at least one published task.

9)Operations by **technicians** on **tasks**:

* List their tasks and show their details.
* Create, update, delete, and publish tasks. Note that published tasks cannot be updated or deleted.

For this deliverable, the task was to implement all the functionality related to the two entities accessible by technicians, as well as the relationships between them (the intermediate table). Throughout the development of this deliverable, I realized—thanks to messages posted in the forum—that the client's requirements had been updated. Additionally, the feedback provided during the first call allowed me to successfully fulfill all of the client's requirements:

**Imagen que contiene Texto

El contenido generado por IA puede ser incorrecto.**

**Texto

El contenido generado por IA puede ser incorrecto.**

The task was to ensure that technicians have appropriate access and views over the two entities they interact with, as well as their relationships (the intermediate table).

**Alternative 1**: Implement two separate views per entity and enforce ownership and publication constraints

Technicians now have access to two tabs per entity:

* One to view all published entities (whether created by them or by other technicians)
* Another to view only the entities they have personally created

When a technician is not the owner of a published entity:

* The entity cannot be edited or deleted, as it is already published
* The technician does not have the permissions to perform actions on it

These constraints apply to both the Record and the Task entities.

Regarding the intermediate entity (representing the relationship between records and tasks):

* Technicians are only allowed to create associations involving records that:
  + They own
  + And that are not yet published, since published records can no longer be modified
* The tasks associated with these records must either:
  + Belong to the technician
  + Or be published tasks created by other technicians (as technicians are only allowed to access tasks that are not their own if they are published)

This implementation ensures that access rights, ownership, and publication status are consistently enforced across all interactions.

**D04:**  
**Produce a test suite for Requirements #8 and #9**  
The task is to create test cases based on the methodologies covered in the course for individual requirements #8 and #9, which involve the functionalities of the entities **Task**, **MaintenanceRecord**, and **InvolvedIn**.

**Alternative 1:**  
Develop the test cases by distinguishing between .safe and .hack files as described in the course materials, providing a wide range of data and achieving full and appropriate data coverage.

* For .safe files, tests must follow the guidelines provided in the theoretical material. Only one file per test is needed if it achieves adequate data coverage.
* For .hack files, tests must also follow the theoretical guidelines, with one file per possible hack case, clearly differentiated in the filename according to the type of hack tested.

**Pros:**

* Tests are created with complete data coverage as specified in the course materials.
* The implemented constraints are properly verified.
* A clear view of all the test cases addressed is provided.
* The different hack cases tested in the system are distinguished.

**Cons:**

* Intentionally left blank.

After analyzing the options, it was decided to use **Alternative 1** as the solution.

**10) Produce a testing report**  
The task is to create a testing report.

**Alternative 1:**  
Create the report following the structure provided in the **Annexes document** given by the faculty, and rely on the theoretical material for the content of the performance testing section.

**Pros:**

* A clear and appropriate structure is followed for creating the report.
* There is a clear overview of the required content, with a distinction between functional and performance tests.

**Cons:**

* Intentionally left blank.

After the analysis, **Alternative 1** was chosen as the solution.

# Conclusions

This report includes an analysis of the requirements requested for the fourth deliverable of the course. It is concluded that all the requested requirements were fulfilled, although with some complications, especially in the system testing phase, as it was laborious to ensure that everything was properly tested with maximum coverage for each operation. However, it proved to be very useful for verifying the proper functioning of the system and for learning new tools.

# **Bibliography**

Intentionally in blank.