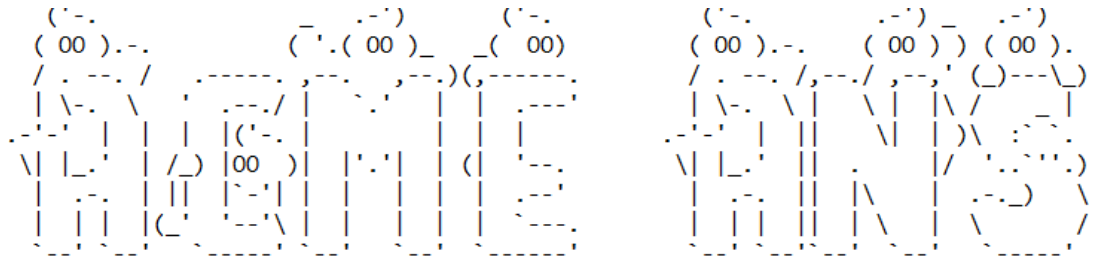
REPORTE DE ANÁLISIS

Acme-ANS-D03



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

Creado por el grupo C1.022, del G1

Participantes	
Nombres	Correos
Fernández Noguerol, Francisco	frafernog@alum.us.es

3 de abril de 2025

Índice

Resumen ejecutivo

En este documento se describe toda la información relacionada con las decisiones de diseño tomadas por el estudiante 1, que en este caso es Francisco Fernández Noguerol, así como las soluciones posibles y las verdaderamente tomadas. La mayoría de decisiones de diseño han sido tomadas basándose en los propios requisitos y su naturaleza para poder completarse.

Tabla de revisiones

Número	Fecha	Descripción
v1.0.0	03/04/2025	Versión finalizada del documento para el entregable 3

Introducción

Para el contenido de este documento, se han tenido en cuenta todas las decisiones de diseño correspondientes a cada uno de los requisitos del entregable 3, donde se describe, mayoritariamente, decisiones de diseño correspondientes a los requisitos de información, ya que se han tenido que modificar bastante con respecto a la entrega anterior, además de alguna decisión tomada para mostrar los datos.

Contenidos

Decisión #1:

Requisito asociado: Requisito 8 / *Operations by managers on their flights:*

- *List the flights that they have created and show their details.*
- *Create, update, or delete their flights. Flights can be updated or deleted as long as they have not been published. For a flight to be published, it must have at least one leg, and all its legs must have been published.*

Problema encontrado: No saber qué relaciones mantener para la clase Flight, ya que esta tenía una manyToMany hacia Leg y no tenía una relación hacia manager.

Soluciones posibles valoradas:

1. Dejarlo estar
    a. Pros:
    Coherencia del código
    Simplicidad a la hora de calcular atributos derivados
    b. Contras:
    Dado un flight sin legs, no se puede saber el manager que lo ha creado
    El framework no permite gestionar relaciones -toMany
2. Dar la vuelta a la navegabilidad entre Leg y Flight y establecer una nueva relación con manager
    a. Pros:
    El framework sí permite estas relaciones y se hace más fácil imprimir los datos por pantalla
    Se mantiene coherencia con las creaciones de flights
    b. Contras:
    Menos legibilidad del código
    Uso de métodos poco recomendados en la entidad (métodos de un repositorio)

Solución adoptada: Dar la vuelta a la relación entre Flight y Leg además de establecer una nueva relación entre Flight y manager.

Validación del profesor: En clases prácticas como teóricas

Decisión #2:

Requisito asociado: Requisito 15 / The system must handle manager dashboards with the following indicators:

- The ranking the manager achieves based on their years of experience. The more years of experience, the higher the position in the ranking.
- The number of years to retire, assuming that they retire at 65.
- Ratio of on-time and delayed legs.
- The most popular and less popular airports within their flights. An airport is popular as long as it has been an origin or destination for many flights.
- The number of legs of their flights grouped by their status.
- The average, minimum, maximum, and standard deviation of the cost of their flights.

Problema encontrado: No poder enviar datos en formatos “complejos” de Java como Map, List, etc.

Única solución encontrada y adoptada: Dividir aquellos valores que a priori se pudieran modelar como clases más avanzadas de Java en más atributos pero de clases más simples como String, Money, etc.

Validación del profesor: Sin validación necesaria.

Decisión #3:

Requisito asociado: Requisito 26 / *The system must track weather conditions. A web service must be used to populate this entity with information about weather conditions. Thus, the exact data to store depends on the chosen service, and it is the students' responsibility to define them accordingly. It is also the students' responsibility to find the appropriate service; no implicit or explicit liabilities shall be covered by the University of Seville or their in-dividual affiliates if the students contract pay-per-use services! The students are strongly advised to ensure that the service they choose is free of charge.*

Problema encontrado: La colección de APIs utilizada en Acme Jobs tenía una cuota muy pequeña para trackear el tiempo

Soluciones posibles valoradas:

1. Usar la API de la página que usa Acme Jobs
    a. Pros:
    Zona de APIs testeada por los profesores
    b. Contras:
    Posible caída del sistema durante su testeo o implementación
2. Buscar otra API más flexible
    a. Pros:
    Mayor versatilidad y libertad a la hora de obtener datos
    b. Contras:
    Encontrar APIs no compatibles con el requisito

Solución adoptada: Buscar una API más flexible, que permite unas 1000 llamadas diarias y que permite obtener los datos requeridos sin tener que estar mirando la cuota.

Validación del profesor: Sin validación necesaria.

Decisión #4:

Requisito asociado: Requisito 30 / *30) Operations by administrators on weather conditions:*

- *Populate the database with forecast weather conditions.*

Problema encontrado: Cómo popular la base de datos con el tiempo.

Soluciones posibles valoradas:

1. Popular los datos introduciendo los datos de las ciudades manualmente
    a. Pros:
    Mayor control del poblado
    Muestreo de errores específicos
    b. Contras:
    Acción muy lenta en caso de querer introducir muchos datos
    Mayor cantidad de clases necesarias para funcionar
2. Popular los datos de todas las ciudades de los aeropuertos pero mostrando las ciudades antes
    a. Pros:
    Mayor rapidez a la hora de querer insertar nuevos datos
    b. Contras:
    Posible acaparación de la cuota de la API en una sola petición
    No mostrar errores sobre qué ciudades no se ha podido obtener el tiempo
    Mayor cantidad de clases necesarias para funcionar
3. Popular los datos de todas las ciudades de los aeropuertos usando como base el “Populate DB” que ya venía en el proyecto base
    a. Pros:
    Mayor rapidez a la hora de querer insertar nuevos datos
    Una única clase para hacerlo funcionar, es decir, un único punto de fallo
    b. Contras:
    Posible acaparación de la cuota de la API en una sola petición
    No mostrar errores sobre qué ciudades no se ha podido obtener el tiempo

Solución adoptada: Seguir el formato del “Populate DB” del proyecto.

Validación del profesor: Sin validación necesaria.

Conclusiones

Como conclusión, en este entregable me han surgido ciertas dudas acerca del modelado de los datos de la aplicación, ya que no sabía si mantener la estructura del entregable anterior o no. Tal y como se explicó en el reporte de análisis de la entrega anterior, estas estructuras se podían ver modificadas, y así ha sido, tanto por restricciones del framework como restricciones implícitos de los propios requisitos.

Bibliografía

Intencionalmente en blanco