INFORME DE ANÁLISIS

Grupo 1-C1.020 | https://github.com/PDJ6975/Acme-ANS-D02-25.2.0

Nombre	Correo Corporativo
Antonio Rodríguez Calderón	antrodcal@alum.us.es
Adrián Ramírez Gil	adrramgil@alum.us.es
Jianwu Hu	jiahu3@alum.us.es
Pablo Castrillón Mora	pabcasmor1@alum.us.es
Pablo Olivencia Moreno	pabolimor@alum.us.es

20 DE FEBRERO DE 2025

PABLO OLIVENCIA | STUDENT #2

Tabla de Contenido

1. Resumen Ejecutivo 1
2. Tabla de Revisión 3
3. Introducción 3
4. Registros de Análisis 3
4.1 Entregable D01 3
4.2 Entregable D02 4
4.3 Entregable D03 6
4.2 Entregable D04 11
5. Conclusión 13
6. Bibliografía 13

1. Resumen Ejecutivo

Este informe detalla los análisis realizados sobre ciertos **requisitos individuales** del proyecto, abordando las **dudas** que surgieron durante su desarrollo, las **posibles soluciones** evaluadas y las **decisiones** finales adoptadas.

A través de este informe, se busca reflejar el proceso llevado a cabo para garantizar el cumplimiento adecuado de los requisitos, minimizando ambigüedades y asegurando la coherencia con la metodología de trabajo establecida.

2. Tabla de Revisión

Versión	Fecha	Descripción de los cambios
1.0	20/02/2025	Creación del documento
2.0	24/03/2025	Actualización D02
3.0	30/03/2025	Actualización D03
4.0	25/05/2025	Actualización D04

3. Introducción

Durante el desarrollo del proyecto, algunos requisitos individuales plantearon dudas sobre su correcta implementación y documentación. Este informe tiene como objetivo exponer dichas dudas, analizar las alternativas consideradas y justificar la elección final de la solución adoptada.

Cada análisis presentado sigue una estructura clara: primero, se expone el requisito específico que generó la duda; luego, se describen las opciones contempladas para abordarlo, analizando ventajas e inconvenientes; finalmente, se detalla la decisión tomada y su justificación (generalmente en base a la respuesta del profesor).

El análisis incluye aspectos como la ubicación adecuada de ciertos elementos dentro de la documentación del proyecto y la diferencia entre enfoques de informes individuales y grupales, cuestiones que fueron aclaradas a través de la consulta directa con el profesor en clase de laboratorio. Este documento busca proporcionar una visión clara del proceso de toma de decisiones.

4. Registros de Análisis

4.1 Entregable D01

Para este primer entregable, no se han presentado problemas que requieran un análisis de posibles soluciones. Se ha creado el documento como constancia de ello y como plantilla para futuras iteraciones.

4.2 Entregable D02

Decisión de Diseño 1: Modelado de BannedPassenger

Opción 1: Entidad BannedPassenger independiente

- **Definición:**
  Se crea una entidad independiente que incluye todos los atributos relevantes del pasajero (por ejemplo, fullName, dateOfBirth, passportNumber) junto con los atributos específicos del baneo (nationality, reason, banDate, liftDate).
- **Ventajas:**
  - **Integridad Histórica:** Se conserva de forma inmutable la información del pasajero en el momento del baneo, evitando que cambios posteriores en la entidad Passenger afecten el registro del baneo.
  - **Auditoría y Trazabilidad:** Permite un seguimiento claro y preciso del estado del baneo, útil para auditorías o consultas históricas.
- **Desventajas:**
  - **Duplicación de Datos:** Se replican ciertos atributos ya presentes en Passenger, lo que puede generar redundancia en el modelo de datos.

Opción 2: Relación entre BannedPassenger y Passenger (por ejemplo, ManyToOne)

- **Definición:**
  Se modela el baneo como una relación con la entidad Passenger, derivando algunos atributos del pasajero relacionado.
- **Ventajas:**
  - **Evita la Duplicación de Datos:** Se utiliza la información actualizada de Passenger sin necesidad de replicarla en otra entidad.
- **Desventajas:**
  - **Pérdida de Información Histórica:** Si los datos del pasajero cambian con el tiempo, el registro del baneo también se actualizará, lo que podría dificultar el seguimiento de la situación en el momento del baneo.
  - **Dependencia de la Entidad Original:** El registro del baneo queda

fuertemente acoplado a la entidad Passenger, lo que puede no ser deseable en ciertos contextos (por ejemplo, auditorías).

Elección Justificada: Entidad BannedPassenger Independiente

- **Preservación Histórica:** Al almacenar de forma independiente los datos del pasajero en el momento del baneo, se asegura que el registro se mantenga inalterable incluso si la información en Passenger cambia posteriormente.
- **Mejor Trazabilidad:** La entidad independiente permite un seguimiento claro y confiable de cuándo y por qué un pasajero fue baneado, así como de si y cuándo se levantó el baneo.
- **Cumplimiento de Requisitos:** La entidad BannedPassenger cumple con el requisito de modelar todos los atributos solicitados (nombre, fecha de nacimiento, pasaporte, nacionalidad, motivo, fecha de baneo y fecha de levantamiento opcional) de forma completa y autocontenida.

Esta decisión favorece la integridad y la trazabilidad de la información en el sistema, asegurando que los registros históricos no se vean afectados por futuras modificaciones en la entidad Passenger.

Decisión de Diseño 2: Modelado de atributos estadísticos en CustomerDashboard

1) **Incluir atributos directamente en la clase CustomerDashboard**
   - **Definición:**

     Los atributos estadísticos (como bookingCostAverage, bookingCostMinimum, passengerCount, etc.) se incluyen directamente en la clase CustomerDashboard en lugar de crear una clase específica para estadísticas.

     Ventajas:

     - **Sencillez:** Evita crear una clase adicional que podría resultar innecesaria si no se va a reutilizar en otro contexto.
     - **Claridad:** Todos los datos relevantes del dashboard están definidos claramente en un único lugar.

     Desventajas:

     - **Poca Reusabilidad:** Si en el futuro fuera necesario reutilizar estas estadísticas, sería necesario refactorizar el código.

- **Mayor Complejidad de la clase:** Puede incrementar la complejidad visual o cognitiva de la clase al incluir muchos atributos relacionados.

2) Crear una clase específica Statistics

- **Definición:**

  Crear una clase separada Statistics que agrupe los atributos estadísticos y reutilizar esta clase dentro del CustomerDashboard.

- **Ventajas:**

  - **Reusabilidad:** Permite reutilizar fácilmente estas estadísticas en otros contextos o dashboards futuros.
  - **Mejor Modularidad:** Facilita la gestión y mantenimiento del código.
- **Desventajas:**
- **=**
  - **Complejidad adicional:** Se crea una clase más, que puede ser innecesaria si no se reutilizan estos atributos en ningún otro sitio.

Elección Justificada: Atributos Directos en CustomerDashboard

- **Contexto Específico:** Debido a que estos atributos no serán reutilizados en otros contextos, la inclusión directa en la clase actual simplifica el modelo y reduce la complejidad.
- **Simplicidad del diseño:** Mantener los atributos estadísticos en la misma clase contribuye a mantener un diseño sencillo y comprensible.

Esta decisión se basa en el principio de simplicidad y en la evaluación de la necesidad real del proyecto, que no anticipa el uso futuro de estos atributos fuera del contexto inmediato del dashboard actual.

4.3 Entregable D03

Decisión de Diseño 1: Gestión de Pasajeros a través de Bookings

1) Acceso a pasajeros desde el menú principal

Definición

Crear opciones de menú separadas para la gestión de pasajeros (listar, crear, editar, eliminar), tratándolos como entidades independientes.

Ventajas:

- Acceso directo a todos los pasajeros desde un único punto.
- Mayor visibilidad de la funcionalidad en la interfaz.

Desventajas:

- Desvinculación contextual entre pasajeros y las reservas a las que pertenecen.
- Posible confusión sobre a qué reserva asignar nuevos pasajeros.
- Mayor complejidad en la interfaz de usuario con más elementos de menú.

2) Gestión de pasajeros integrada en la vista de reservas

Definición:

Implementar la gestión de pasajeros como una funcionalidad subordinada a las reservas, accesible solo dentro del contexto de una reserva específica.

Ventajas:

- Mantiene el contexto: los pasajeros siempre se visualizan/gestionan en relación con su reserva.
- Flujo de trabajo más natural: primero se crea la reserva, luego se añaden los pasajeros.
- Seguridad mejorada: solo el propietario de la reserva puede gestionar sus pasajeros.

Desventajas:

- No hay una vista global de todos los pasajeros del sistema.
- Requiere navegación adicional para llegar a la gestión de pasajeros.
- Elección Justificada: Gestión de pasajeros integrada en reservas
- Integridad contextual: Los pasajeros solo tienen sentido en el contexto de una reserva específica. La relación entre Passenger y Booking a través de BookingRecord refuerza esta dependencia.

- Flujo de usuario natural: Refleja el proceso real de reserva: primero se crea la reserva para un vuelo y luego se añaden los pasajeros, no al revés.

- Seguridad y control de acceso: La autorización se implementa más fácilmente al comprobar super.getRequest().getPrincipal().hasRealm(booking.getCustomer()) dentro del contexto de una reserva específica.

- Simplificación del modelo mental: Para el usuario, los pasajeros son "parte de" una reserva, no entidades independientes, lo que hace que la aplicación sea más intuitiva.

Decisión de Diseño 2: Limitaciones de llamadas a la API de Foursquare Places

1) Implementar límites adicionales en la recuperación de recomendaciones

Definición:

Establecer un límite explícito en el número de recomendaciones recuperadas por aeropuerto, más allá de las restricciones ya configuradas en la API (por ejemplo, limitar a 10 recomendaciones por aeropuerto).

Ventajas:

- Control más granular sobre la cantidad de datos almacenados en la base de datos.
- Reducción del tiempo de procesamiento y del tamaño de las respuestas de la API.
- Menor uso de recursos del sistema (almacenamiento y procesamiento).

Desventajas:

- Posible pérdida de información relevante si el límite es demasiado restrictivo.
- Requiere lógica adicional para filtrar y priorizar las recomendaciones más relevantes.
- Introduce una capa de complejidad innecesaria si la API ya tiene un límite configurado.

2) No establecer límites adicionales en la recuperación de recomendaciones

Definición:

Permitir que el sistema recupere todas las recomendaciones disponibles dentro de los límites establecidos por la API (por ejemplo, 5 recomendaciones por consulta, como se configura en el parámetro limit).

Ventajas:

- Cumple con los requisitos del proyecto, que especifican un enfoque proactivo por parte de un administrador sin restricciones adicionales.
- Aprovecha al máximo las capacidades de la API para proporcionar información completa y relevante.
- Simplifica la implementación al delegar la gestión de límites a la API externa.

- La API de Foursquare Places ofrece un número muy grande de llamadas, lo que permite realizar consultas sin preocuparse por exceder los límites en el entorno actual.

Desventajas:

- Puede generar un mayor volumen de datos almacenados en la base de datos si la API devuelve muchas recomendaciones.
- Posible sobrecarga en el sistema si se procesan demasiados datos en una sola ejecución.
- Elección Justificada: No establecer límites adicionales en la recuperación de recomendaciones
- Cumplimiento de requisitos: El requisito explícito del proyecto establece que el mecanismo debe ser proactivo y administrado manualmente por un administrador, sin necesidad de imponer límites adicionales más allá de los configurados en la API.
- Uso eficiente de la API: La API de Foursquare Places ya tiene un límite configurado en el parámetro limit = "5", lo que asegura que no se exceda un número razonable de recomendaciones por consulta. Además, la API ofrece un número muy grande de llamadas, lo que permite realizar consultas sin preocuparse por exceder los límites en el entorno actual.
- Filtrado inteligente en el controlador: En el método doPopulateRecommendations() del controlador AdminRecommendationPopulateController, solo se realizan llamadas a la API para aeropuertos con ciudades válidas (aquellas que no contienen "lorem ipsum"). Esto asegura que las consultas se enfoquen en datos relevantes, ya que la mayoría de las ciudades en el entorno de muestra actual contienen "lorem ipsum".
- Simplicidad en la implementación: Al no imponer límites adicionales, el sistema se mantiene más simple y delega la responsabilidad de la gestión de resultados a la API externa, que ya está optimizada para este propósito.
- Cobertura completa de datos: Permitir que la API devuelva todas las recomendaciones disponibles dentro de sus propios límites garantiza que los usuarios tengan acceso a la mayor cantidad de información relevante posible.
- Escalabilidad: Este enfoque permite que el sistema se adapte fácilmente a diferentes volúmenes de datos sin necesidad de ajustes adicionales, ya que la API controla la cantidad de resultados devueltos.

Decisión de Diseño 3: Formato de visualización de vuelos en la creación de bookings

1) Mostrar solo información básica del vuelo (ID o código)

Definición:

Presentar los vuelos usando únicamente identificadores o códigos en los selectores.

Ventajas:

- Implementación simple.
- Interfaz más compacta.

Desventajas:

- Poca información para que el usuario tome decisiones.
- Necesidad de navegación adicional para ver detalles del vuelo.

2) Visualización enriquecida con información contextual

Definición:

Mostrar información compuesta y contextual de cada vuelo que incluya origen, destino y otra información relevante.

Ventajas:

- Más información disponible en el punto de decisión.
- Reducción de pasos en el proceso de reserva.
- Mejor experiencia de usuario con información contextualmente relevante.

Desventajas:

- Implementación más compleja.
- Posible sobrecarga visual si hay muchos atributos.

Elección Justificada: Visualización enriquecida con información contextual

- Mejora en la toma de decisiones: Como se ve en CustomerBookingCreateService.unbind(), se construye una etiqueta informativa que incluye el tag del vuelo y las ciudades de origen y destino (tag + " (" + flight.getOriginCity() + " → " + flight.getDestinationCity() + ")"), proporcionando información crucial para que el usuario elija adecuadamente.

- Experiencia de usuario optimizada: Al presentar información esencial en

el mismo selector (origen, destino, identificador), se elimina la necesidad de consultar detalles adicionales del vuelo en otra pantalla.

- Consistencia con flujos reales de reserva: Simula las interfaces de las aerolíneas del mundo real, donde se muestran datos clave del vuelo (origen, destino, horarios) durante el proceso de reserva.

- Eficiencia en el proceso: Reduce el número de pasos necesarios para completar una reserva, mejorando la eficiencia y satisfacción del usuario.

4.2 Entregable D04

Para este entregable quería detallar dos decisiones de diseño tomadas en anteriores entregables, pero no correctamente detalladas.

Decisión de diseño 1: Atributo Price en Booking

1. **Precio calculado automáticamente (opción descartada)**

Definición:

Calcular el precio del Booking multiplicando el coste del vuelo por el número de pasajeros.

Ventajas:

- Automatización del proceso de reserva.
- Coherencia entre los precios definidos en los vuelos y las reservas.
- Menor margen de error por parte del usuario.

Desventajas:

- Requiere definir claramente el significado del atributo "cost" del vuelo, lo cual no estaba suficientemente especificado en los requisitos.
- Aumenta la complejidad de la lógica en BookingCreateService.
- No queda claro si todos los pasajeros deberían tener el mismo precio, o si hubiera factores adicionales a considerar.

2. **Precio introducido manualmente por el usuario (opción elegida)**

Definición:

El usuario debe introducir de forma explícita el precio total del Booking mediante un campo obligatorio.

Ventajas:

- Implementación más sencilla.
- Flexibilidad para el usuario.
- Aceptado por el profesorado durante sesiones de revisión.

Desventajas:

- Menor integridad referencial respecto al coste del vuelo.
- Menos realista para un sistema comercial.

Elección Justificada: Precio introducido manualmente por el usuario.

Se decidió adoptar esta opción por su simplicidad, y porque la ambigüedad en la definición del atributo "cost" del vuelo no permitía implementar un sistema de cálculo fiable. Además, se confirmó con el profesorado que este enfoque era aceptable para los fines del proyecto.

Decisión de diseño 2: Atributo Flight en read-only

1. **Permitir modificar el vuelo asociado a una reserva existente (opción descartada)**

Definición:

El campo "Flight" sería editable en la operación de actualización de reservas.
Ventajas:

- Flexibilidad para el usuario.
- Posibilidad de corregir errores o cambiar planes.

Desventajas:

- Modificar el vuelo podría invalidar lógicas dependientes, como el precio, la disponibilidad, o los registros asociados.
- En un sistema real, cambiar el vuelo de una reserva suele implicar una nueva reserva, no una actualización.
- Mayor complejidad para validar la coherencia de los datos.

2. **Dejar el campo Flight en modo read-only (opción elegida)**

Definición:

No permitir editar el vuelo una vez creado el Booking. El atributo se visualiza pero no se puede cambiar.

Ventajas:

- Sencillez de implementación.
- Evita inconsistencias y errores lógicos.
- Coherente con el flujo de reserva: si quieres otro vuelo, creas otra reserva.

Desventajas:

- Menor flexibilidad para el usuario final.

Elección Justificada: Dejar el campo Flight en read-only.

Se considera más apropiado y realista que una reserva, una vez creada para un vuelo determinado, no permita cambiar ese vuelo. Esta decisión también simplifica la implementación y evita errores que podrían surgir de modificaciones inconsistentes del modelo.

5. Conclusión

Este informe ha documentado las dudas surgidas durante el desarrollo del proyecto, analizando distintas alternativas y justificando las decisiones tomadas en cada caso. Se ha seguido un enfoque estructurado para evaluar las opciones disponibles y validar la solución más adecuada en cada situación.

El análisis realizado permite garantizar una interpretación precisa de los requisitos y una correcta aplicación de los criterios establecidos. Además, contribuye a mejorar la toma de decisiones y optimizar el proceso de desarrollo, asegurando coherencia y alineación con los objetivos del proyecto.

6. Bibliografía

Intencionalmente en blanco.