INFORME DE ANÁLISIS INDIVIDUAL

Grupo 1-C1.020 | https://github.com/PDJ6975/Acme-ANS-D03-25.3.0

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

18 DE ABRIL DE 2025

ANTONIO RODRÍGUEZ | STUDENT #3

Tabla de Contenido

1. Resumen Ejecutivo

Este informe detalla los análisis realizados sobre los requisitos individuales del estudiante número tres 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	18/02/2025	Creación inicial del documento
1.1	19/02/2025	Corregir números de contenido
2.0	28/02/2025	Actualizar contenido para el segundo entregable
2.1	09/03/2025	Introducir más decisiones
3.0	18/04/2025	Actualizar contenido para tercer y quinto entregable

3. Introducción

Durante el desarrollo del proyecto, algunos requisitos individuales crearon dudas sobre su correcta implementación y documentación. Este informe tiene como objetivo comentar 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).

4. Registros de Análisis

En este apartado se aborda en primera persona el análisis introducido en los apartados anteriores de forma completa.

4.1 Relacionados con la primera entrega

4.1.1 Requisito 2:

Provide a link to your planning dashboard in GitHub to review the tasks, their current status, and your schedule

Al leer este requisito, me surgió la duda sobre dónde debía colocar el enlace solicitado. Al revisar el documento **“On Your Deliverables”**, observé que el espacio debajo del requisito solo debía usarse para indicar con una "X" si estaba completado o no, pero no se especificaba dónde debía incluirse el enlace. A partir de esto, consideré dos opciones posibles:

Opción 1: Colocar el enlace en la entrega de Enseñanza Virtual.

Pensé en incluirlo en el campo de la entrega donde se adjunta el proyecto. Sin embargo, el documento de instrucciones no menciona esta posibilidad explícitamente, por lo que no estaba seguro de si sería la mejor opción.

Opción 2: Incluir el enlace en el fichero “README” del proyecto.

En un principio, esta opción me generaba dudas, ya que no veía claro que el README fuera el lugar adecuado para esta información. Además, no estaba seguro de si los profesores iban a revisar este fichero.

Para resolver esta pregunta, en una sesión de seguimiento de laboratorio, un compañero planteó esta misma duda al profesor. Su respuesta confirmó que colocar el enlace en el README era una opción válida.

Conclusión

Finalmente, decidí incluir el enlace en el README del proyecto, ya que cumple con el requisito y fue validado como una solución correcta en clase.

4.2.2 Requisito 24/25:

Produce an analysis report / Produce a planning and progress report

Cuando leí estos requisitos, me surgió la duda de cómo debía abordarlos, ya que ambos estaban definidos tanto como requisitos individuales suplementarios como requisitos grupales suplementarios. No tenía claro cuál era la diferencia entre ambas versiones ni cómo debía enfocarlos en cada caso.

A partir de esto, consideré dos posibles interpretaciones:

Opción 1: Tomar una versión global completa en el informe grupal y una individual completa para los individuales

En el informe grupal se debían tratar todos los requisitos del proyecto, tanto grupales como individuales, realizados por todos los miembros del equipo. Mientras que en el informe individual debía hablar únicamente de los requisitos grupales e individuales en los que yo hubiera trabajado personalmente.

Opción 2: Realizar una distinción clara entre grupales e individuales

En el informe grupal solo debían abordarse los requisitos grupales, mientras que en el informe individual solo se debían tratar los requisitos individuales propios de cada miembro.

Mi duda principal era si los miembros del equipo debíamos considerar los requisitos individuales de los demás, o si cada uno era responsable de su parte individual.

Para aclararlo, planteé la cuestión al profesor en una clase de laboratorio, y su respuesta fue que nos podíamos organizar como quisiéramos, pero que los requisitos individuales eran responsabilidad exclusiva de cada persona. Así, una buena manera de estructurar los informes sería siguiendo la Opción 2.

Conclusión

Es por ello por lo que al final me decanté por seguir la segunda opción, realizando una clara separación entre los tipos de requisitos.

4.2 Relacionados con la segunda entrega

4.2.1 Requisito 5:

*An **activity log** records incidents that occur during a flight. They are logged by any of the **flight crew members** assigned to the corresponding leg and after the **leg** has taken place. The incidents include weather-related disruptions, route deviations, passenger issues, or mechanical failures, to mention a few. Each log entry includes a **registration moment** (in the past), a **type of incident** (up to 50 characters) a **description** (up to 255 characters), and a **severity level** (ranging from 0 to 10, where 0 indicates no issue and 10 represents a highly critical situation).*

La duda con este requisito está relacionada principalmente con la necesidad de crear la entidad “ActivityLog (nombre de ejemplo)”. Como se menciona, una etapa de vuelo va a tener asociado un registro de actividad, de manera que cada entrada del mismo tendrá asociado un incidente registrado por cualquier miembro de la tripulación asignado en esa etapa.

Bien, el problema viene en que entonces un registro de actividad no es más que un listado de incidentes, de manera que se puede cuestionar la existencia de la entidad.

A partir de esto, consideré dos posibles interpretaciones:

Opción 1: Mantener la existencia de la entidad "Activity Log".

En esta opción se busca mantener la entidad, de forma que desde ella se puede acceder unidireccionalmente a la etapa de vuelo "FlightStage" mediante una relación @OneToOne. Del mismo modo, un registro tiene una relación con un listado de incidentes, pero se accede unidireccionalmente desde los incidentes mediante una relación @ManyToOne.

> Esta opción puede generar el problema de que al final tienes una entidad prácticamente vacía y con una relación @OneToOne que no es muy utilizada en la asignatura, ya que se insiste mucho en las relaciones @ManyToOne.

Opción 2: Eliminar la entidad "Activity Log"

La otra opción contemplada sería eliminar la entidad y mantener una relación directa @ManyToOne entre "FlightStage" y la lista de incidentes (cada uno es una entidad "LogEntry"). En un principio, parece la solución más viable ya que estás eliminando una entidad y una relación que en un principio no son necesarias.

Conclusión

Al final, me decanté por utilizar la segunda opción. Así, nos va a quedar una relación @ManyToOne directa y un diagrama UML más limpio.

4.2.2 Requisito 26:

*The system is required to provide crew members with information about **visa requirements**. A web service must be used to populate this entity with information about visa requirements. 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 individual affiliates if the students hire pay-per-use services! The students are strongly advised to ensure that the service they choose is free of charge.*

La duda con este requisito ha sido principalmente con qué API decantarme, ya que había encontrado dos opciones bastante viables, una API abierta de GitHub gratuita y una API con un plan gratuito de 600 peticiones mensuales.

A partir de esto, consideré dos posibles interpretaciones:

Opción 1: Utilizar la API gratuita.

El problema de esta opción es que la API devolvía muy poca información sobre los requisitos de visa, generalmente indicando solo el tipo de visa necesario, lo que podía ser insuficiente para cumplir con la tarea, pues al final tendríamos una sola entidad con uno o dos atributos a almacenar.

Opción 2: Utilizar la API de pago con el plan gratuito (Travel Buddy)

La otra opción contemplada sería utilizar la API de pago con el plan gratuito de 600 peticiones. Lo bueno de esta opción es que la API devolvía mucha más información relevante e interesante para cumplir con la tarea. Me preocupaba que consumiese más de una petición por llamada, pero no era así.

Conclusión

Al final, me decanté por utilizar la segunda opción, pues pienso que 600 peticiones al mes son más que suficientes para cumplir con la tarea de la asignatura sin llegar a su límite, manteniendo así una tarea interesante.

4.2.3 Requisito 6:

*Produce assorted sample data to test your application informally. The data must include four **flight crew member** accounts with credentials "**memberX/memberX**" with X ranging from 1 to 4 (and different duties each). Additionally, create a fifth member account **member/member**, representing a new member with no flight assignment, as if the account had just been created.*

La duda principal en este requisito ha sido al poblar el rol "FlightCrewMember", pues en la tarea se exige crear cinco cuentas de usuario para estos miembros en total. El problema está en que no se habla de poder incluir más cuentas para este rol, de forma que el número de filas en el csv de "FlightCrewMember" puede verse limitado.

Si bien la aplicación permite como tal que una cuenta de usuario pueda estar asociada a más de una instancia de un rol por una relación @ManyToOne, es algo que carece de sentido en la práctica.

Así, si se mantienen solo las cinco cuentas, podríamos crear cinco filas como máximo en el csv de FlightCrewMember, en donde asociemos un miembro a cada cuenta, por lo que no se puede probar de forma exacta y completa todos los atributos de la entidad puesto que el tamaño limitado no permite probar todas las variaciones.

A partir de esto, consideré dos posibles interpretaciones:

Opción 1: Asociar cada cuenta de usuario con más de un miembro

El problema de esta opción es que en la práctica carece de sentido, aunque la ventaja es que se podría probar de forma completa la cobertura de la entidad.

Opción 2: Asociar cada cuenta a un solo miembro

Si bien con esta opción la cobertura disminuye, seguiría el ejemplo de roles establecidos en Acme Job como “Employer”, que asocia un solo empleado a cada cuenta.

Conclusión

Al final, me decanté por utilizar la segunda opción, pues creo que basarse en Acme Job es lo más viable, y si bien la cobertura puede disminuir, en tan solo cinco filas creo que se pueda abordar de forma bastante completa todas las variaciones, puesto que la entidad tiene pocos atributos que verdaderamente haya que probar.

4.2.4 Requisito 4:

*A **flight assignment** represents the allocation of a **flight crew member** to a specific **leg** of a flight. Each assignment specifies the flight crew's **duty** in that leg ("PILOT", "CO-PILOT", "LEAD ATTENDANT", "CABIN ATTENDANT"), the **moment** of the last update (in the past), the **current status** of the assignment ("CONFIRMED", "PENDING", or "CANCELLED"), and some **remarks** (up to 255 characters), if necessary.*

Duda 1: Cómo debe definirse el atributo remarks

La duda principal con este requisito viene con el atributo “remarks”, puesto que se especifica en plural y por tanto puede hacer dudar de si debe ser una lista de comentarios (lista de String), o un único String.

En este sentido, si se establece como una lista, no se podría validar directamente a través del framework que cada comentario tenga una longitud máxima de 255 caracteres, mientras que si se usa directamente un String sí se podría, utilizando la notación @ValidString.

A partir de esto, consideré dos posibles interpretaciones:

Opción 1: Dejar una lista y de alguna manera en el servicio limitar la longitud de cada comentario al querer crear o editar una asignación de vuelo

El problema de esta opción es que es bastante más compleja que simplemente validar en la entidad y va a quedar un servicio menos limpio.

Opción 2: Dejar el atributo como un único String en el que se utilice la notación @ValidString

Con esta opción el código sería mucho más limpio y sencillo de entender, de forma que se promueve la separación de responsabilidades y la validación a nivel de entidad.

Conclusión

Al final, me decanté por utilizar la segunda opción, pues pienso que es la mejor a la hora de llevar a cabo el desarrollo y creo que cumple con lo que la tarea quiere especificar.

Duda 2: Límites para @ValidString opcionales

Por otra parte, este atributo también ha generado otra duda, y es que, al ser opcional, aunque limites el @ValidString a “min=1”, permitiría la inserción de valores vacíos que no se convertirían a nulo. Esto no ocurre cuando es obligatorio porque la notación directamente lo convertiría a nulo.

A partir de esto, consideré dos posibles interpretaciones:

Opción 1: Introducir algún tipo de notación como @NotBlank o crear un validador personalizado

El problema de esta opción es que puede ser tediosa y no cumplir a la perfección con lo exigido en la inmensa mayoría de entidades que se desarrollan con el framework.

Opción 2: Hacer un setter propio para ese atributo

En vez de dejar que lombok genere un setter para ese atributo, se puede crear uno personalizado de manera que, si el comentario no es nulo y al hacer el "trim" está vacío, se establezca a nulo.

Conclusión

Al final, me decanté por utilizar la segunda opción, pues la más sencilla y va a añadir un bloque de seguridad adicional a la aplicación.

4.3 Relacionados con la tercera/quinta entrega

4.3.1 Requisito 8:

*Operations by **flight crew members** on **flight assignments**:*

- *List the flight assignments separately, one for completed flight legs and another one for those planned but that have not taken place yet.*
- *Show the details of their flight assignments and the associated legs and flight crew members.*
- *Create, update, and publish their flight assignments. Only crew members with duty "LEAD ATTENDANT" can perform these operations. Please, note that to publish a flight assignment these cannot be linked to legs that have already occurred. Additionally, only flight crew members with an "AVAILABLE" status can be assigned to a leg, and they cannot be assigned to multiple legs simultaneously. Lastly, each leg can only have one pilot and one co-pilot. The allocation of remaining roles for other flight crew members is at the discretion of the "LEAD ATTENDANT". Flight assignments can be updated or deleted as long as they have not been published.*

Duda 1: Influencia del atributo draftMode en las asignaciones y etapas

Tanto para la entidad de las asignaciones como para la de las etapas, se especifica como necesario el atributo draftMode para saber cuándo una etapa o asignación está publicada. Así, mi duda surge de la necesidad de que una asignación pueda o no estar vinculada a etapas en modo borrador.

Opción 1: Permitir que las asignaciones puedan estar vinculadas a etapas independientemente de su estado en el sistema

Sinceramente, pienso que esta opción no es la más coherente, pues lo lógico es tener primero una versión estable y fija de una etapa en el sistema (es decir, publicada) para posteriormente ir haciendo asignaciones sobre dicha etapa. No hay ninguna necesidad y/o ventaja en permitir realizar asignaciones a etapas que están en borrador.

Opción 2: Permitir que las asignaciones solo estén vinculadas a etapas publicadas en el sistema

Pienso que esta es la mejor opción si queremos tratar de mantener una consistencia general en el sistema, permitiendo realizar asignaciones una vez sepamos que la etapa no va a cambiar más.

De hecho, es lógico, puesto que desde el punto de vista de un miembro asignado, le interesa que la etapa sobre la que ha tenido la asignación no vaya a sufrir cambios.

Conclusión

Al final, me he decantado por la segunda opción, puesto que creo que desde un punto de vista realista es la opción más normal y segura de cara a asegurar un flujo general consistente.

Así, a la hora de listar asignaciones me aseguro de que solo se recuperen aquellas cuya etapa está publicada, y a la hora de crearlas me aseguro de lo mismo.

Duda 2: Dependencia del rol LEAD_ATTENDANT para la creación, actualización y borrado de asignaciones

La duda en este requisito recaía principalmente en la complejidad y algunas incongruencias a la hora de tener en cuenta el rol LEAD_ATTENDANT en la gestión de asignaciones.

Opción 1: Ceñirnos al requisito y utilizar el rol

El problema de tener en cuenta el rol es que habría estados inconsistentes. Por ejemplo, si se crease una etapa de cero, no se podría realizar ninguna asignación sobre ella porque no habría un LEAD_ATTENDANT que iniciase el proceso.

Por lo demás, el requisito sí se podría llevar a cabo, validando en qué asignaciones el miembro registrado es LEAD_ATTENDANT y, sobre ellas, permitir la creación, borrado y actualización (estas dos últimas si la asignación está en borrador).

Opción 2: No tener en cuenta el rol y permitir autoasignaciones

Esta alternativa me parece la mejor. El miembro registrado en el sistema puede autoasignarse a una etapa siempre que esta sea pública y no haya ocurrido (no tiene sentido hacer asignaciones sobre etapas pasadas).

El único problema que podría surgir con esta opción sería si existiera una restricción en cuanto al número de LEAD_ATTENDANT por etapa. Sin embargo, debido a la solución dada en el foro (que ahora mencionaré en la conclusión), he decidido que puede haber más de un LEAD_ATTENDANT, puesto que no supone un problema para el funcionamiento de la aplicación y es la solución más sencilla.

Conclusión

Al final, si bien en un principio implementé la primera opción, he acabado decantándome por la segunda. Los profesores han indicado en el foro que esta es la solución más adecuada para el contexto del sistema. *(Ver:* enlace_foro1*)*

Duda 3: Qué entendemos por etapa que ha ocurrido

La duda aquí reside en cuándo entendemos que una asignación pertenece a una etapa completada o planeada, ya que disponemos tanto de un atributo de tipo fecha (scheduledArrival) como de un enumerado con el estado de la etapa.

Opción 1: Tener en cuenta solo el enumerado del estado de la etapa

Pienso que no es la mejor opción, ya que basarse en el enumerado puede llevar a inconsistencias en la base de datos. Por ejemplo, podría darse el caso de que una etapa ya haya aterrizado (scheduledArrival < now, fecha del sistema), pero que el estado no se haya actualizado aún a LANDED.

Por tanto, veo este atributo más problemático que beneficioso. De hecho, si una etapa ya está publicada, se entiende que no va a sufrir más retrasos ni modificaciones. En ese caso, scheduledArrival sería un criterio suficiente.

Opción 2: Usar tanto el enumerado como el atributo scheduledArrival

En consonancia con lo anterior, pienso que esta opción solo añadiría más carga al código y no sería necesaria cuando ya contamos con un atributo fiable como scheduledArrival.

Opción 3: Usar solo scheduledArrival para determinar cuándo una etapa se ha completado

Considero que esta es la opción más viable, por todo lo comentado. Una etapa publicada no puede ser modificada, por lo que su scheduledArrival se convierte en una referencia fiable para saber si ha finalizado.

Conclusión

Me he decantado por la tercera opción, siguiendo además la respuesta de los profesores en el foro, donde se menciona explícitamente que no hagamos caso al atributo de estado. *(Ver:* enlace_foro_2*)*

4.3.2 Requisito 9:

*Operations by **flight crew members** on **activity log records**:*

- *List the activity log records in their flight assignments.*
- *Show the details of their activity log records.*
- *Create, update, delete and publish activity log records. They cannot be published if their corresponding flight assignments have not been published yet. No updating or deletion is possible once an activity log record has been published.*

Duda 1: Estado de publicación de las asignaciones para los incidentes

La duda con este requisito también trata sobre el atributo draftMode, puesto que en el requisito se menciona que solo se puede publicar un incidente si la asignación correspondiente ha sido publicada, lo que puede dar a entender que se pueden asociar incidentes en modo borrador a asignaciones en modo borrador.

La duda es si es lógico que se puedan registrar incidentes de asignaciones que no están publicadas en el sistema.

Opción 1: Permitir asociar incidentes a asignaciones en modo borrador y comprobar al publicar el incidente que la asignación ya se ha publicado.

Sinceramente, pienso que esta opción no es la mejor desde un punto de vista de mantener un flujo consistente. Según mi lógica implementada, una asignación se va a mantener firme una vez sea pública en el sistema, al igual que su etapa.

Por tanto, crear incidentes sobre asignaciones que todavía no son públicas carece de sentido, por lo menos desde un punto de vista funcional. Lo mejor es solo asociar incidentes una vez la asignación es pública, confirmada, y la etapa también pública, lo que significa que hay una asignación consistente a la que se van a poder asociar incidentes.

Opción 2: Solo permitir crear incidentes para asignaciones publicadas y confirmadas

Creo que esta es la mejor opción a la hora de crear incidentes, principalmente por el tema de consistencia ya comentado.

A la hora de publicar, se volvería a comprobar que la asignación fuese pública y confirmada, y que la etapa también fuese pública, para evitar que incidentes corruptos puedan publicarse en el sistema.

No obstante, estas validaciones también se implementan a la hora de mostrar el botón de incidentes en las asignaciones, por lo que tenemos una primera capa de seguridad en el propio frontend.

Conclusión

Al final, pienso que lo mejor que se puede hacer es implementar la segunda opción. Todavía no tengo una respuesta oficial del profesorado, pero está pendiente para la próxima sesión de laboratorio.

Duda 2: Para qué etapas es conveniente escribir incidentes

Esta duda recae en si es lógico que se puedan escribir incidentes para asignaciones de etapas que todavía no han terminado, o si es más lógico escribir incidentes solo para asignaciones de etapas ya finalizadas.

Opción 1: Escribir incidentes solo para asignaciones cuando la etapa ha concluido

Desde un punto de vista lógico, parece algo bastante correcto. Generalmente, los miembros van a escribir incidentes cuando la etapa ya haya acabado, salvo que prefieran escribirlos durante el transcurso de la propia etapa. Es por ello que veo más viable la siguiente opción.

Opción 2: Permitir que se escriban incidentes independientemente del estado de la etapa

Pienso que esta es la opción más viable, ya que damos la posibilidad de que un miembro pueda escribir incidentes tanto cuando la etapa está en curso como cuando ya ha finalizado. Creo que es lo mejor desde un punto de vista lógico y también desde la implementación, ya que tendríamos que hacer menos comprobaciones.

Conclusión

Al final, me he decantado por la segunda opción, pues creo que es lo más sencillo y conveniente a la hora de implementarlo. No obstante, todavía no tengo una respuesta oficial del profesorado, pero está pendiente para la próxima sesión de laboratorio.

4.3.3 Requisito 30:

*Operations by **administrators** on **visa requirements**:*

- *Populate the database with visa requirements data*.

Mi duda en este requisito recae en el tema de las **cuotas de la API** que he empleado para obtener los requisitos de visado (Travel Buddy), la cual ya he tratado previamente en este informe.

La API mencionada tiene un **límite de 600 peticiones gratuitas al mes**. Si bien esta cifra puede parecer suficiente, lo cierto es que se puede agotar con bastante facilidad.

Opción 1: No limitar el número de países y cambiar de API

Consistiría en no restringir el número de combinaciones de países al poblar la base de datos, y en su lugar cambiar de API a una totalmente gratuita o con un mayor número de peticiones mensuales.

Sinceramente, pienso que esta opción no es viable. Toda la aplicación (mi parte) ha sido desarrollada en torno al uso específico de Travel Buddy API, y cambiarla podría suponer una gran inversión de tiempo.

El problema que no anticipé fue que, en cuanto se tenga un listado con tan solo diez países, la combinatoria crece rápidamente y el consumo de peticiones a la API se dispara, agotando la cuota en muy poco tiempo.

A pesar de esto, estuve una gran cantidad de tiempo investigando opciones de APIs y esta fue con diferencia la más adecuada en cuanto a calidad, respuesta y documentación.

Opción 2: Limitar la combinatoria y rango de países

Esta opción consiste en mantener el uso de Travel Buddy, pero restringiendo el número de combinaciones a consultar. Se implementa definiendo un listado reducido de combinaciones predefinidas, suficiente para probar la funcionalidad sin comprometer la cuota mensual.

De este modo, se evita el riesgo de superar el límite de peticiones sin necesidad de realizar cambios importantes.

Conclusión

Me he decantado por la segunda opción, ya que es la más viable en términos de tiempo y funcionalidad ofrecida por la API.

En la implementación actual, he limitado la llamada a la API a dos combinaciones, pero tengo comentado en el código hasta veinticinco combinaciones adicionales, que serán activadas en la entrega final.

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.

6. Bibliografía

Intencionalmente en blanco.