**INFORMACIÓN FORO:**

**-Anotaciones generales:**

* **Validación de atributos:** La recomendación que les damos es que todos sus atributos tengan el siguiente esquema:
  + **@Mandatory/@Optional:** Se usan para restricciones de obligatoriedad / opcionalidad de valor para el atributo.
  + **@ValidString/@ValidNumber/@ValidMoment/@ValidMoney/ @ValidScore** **o custom:** Se usan para restricciones de validación.
  + **@Automapped/@Column/@Temporal/@ManyToOne/@OneToOne:** Se usan paramapear sobre la base de datos.

La capa L3 es la de mapeo sobre la BD.  No es ninguna capa de validación.  Sólo es una capa que le dice al framework lo siguiente:

* + **@Automapped:** framework, mapea este atributo como tu sabes; el mapeo por defecto me vale (hemos estudiado esos mapeos por defecto en L02/S03).
  + **@Column(unique = true):** framework, crea en la BD un índice para este atributo y dile a MariaDB que \*falle\* si llegan valores duplicados.
  + **@Temporal(TIMESTAMP):** framework, dile a MariaDN que este atributo almancena una fecha y una hora (un momento) y que debe usar el tipo “datetime”.

Puede Ud. definir un validador a nivel de atributo, pero nosotros le hemos recomendado que lo haga a nivel de clase de entidad, dado que generalmente son varias las restricciones custom que tenemos que implementar y así lo podemos hacer todo en un único validador.

Si se pregunta Ud. para qué definimos ese índice único si realmente lo tenemos que comprobar en la aplicación es por un motivo de eficiencia: para determinar que no hay otro objeto en la BD con el mismo valor tiene Ud. que lanzar una consulta para buscar un objeto con ese valor y dicha consulta requiere de ese índice único para ser eficiente.

-**S05-D02-03:**

* **Technician-YearsOfExperience:** El cliente nos indica que tan sólo le interesa saber el número de años en conjunto; no ve diferencia entre 5.1 y 5.9; este valor tan sólo le da una idea aproximada de la cantidad de experiencia acumulada que tiene una persona.  Por lo tanto se puede entender como un **entero.**
* +----------------------------------------+--------------+-------------+  
  |           Attribute                   | Minimum   | Maximum |  
  +----------------------------------------+--------------+-------------+  
  | Technician::Specialization            |   1 (1)     |   50       |  
  | Technician::Years of experience |   0         |   120 (2) |   
  | Technician::Certifications             |   1 (1)     |   255    |  
  | Technician::License number        |   8 (3)     |   9       |   
  | Technician::Phone number           |   6 (3)     |   15      |  
  | MR::Next inspection... |   \*        |   \*       |  
  | MaintenanceRecord::Notes         |   0 (1)     |   255     |  
  | MR::Maintenance moment |   \*         |   \*       |  
  | MaintenanceRecord::Status       |   - (4)     |   - (4)   |  
  | MR::Estimated cost     |   \*        |  \*       |  
  | Task::Description                     |   1 (1)     |  255     |  
  | Task::type                            |   - (4)     |   - (4)   |  
  | Task::Priority                        |   0        |   10      |  
  | Task::Estimated duration      |   0         |   1000    |  
  +---------------------------------------+---------------+--------------+
* **(1)** Si los requisitos indican que una cadena tiene una longitud máxima de N y es obligatoria, entonces el tamaño mínimo es 1. Si es opcional, entonces el tamaño mínimo es 0.
* **(2)** Recordemos que sólo necesitamos definir rangos suficientemente amplios. Seguramente ningún técnico tendrá 120 años de experiencia, pero ni nosotros ni nuestro cliente podemos establecer un límite superior exacto; sólo sabemos que el mínimo es cero años (alguien que acaba de empezar).
* **(3)** Este atributo tiene un patrón regular obligatorio. En este caso, el patrón define la longitud mínima y máxima.
* **(4)** Este es un atributo con valores enumerados, por lo que no aplica el concepto de mínimo o máximo; el atributo puede tomar cualquiera de los valores definidos en la enumeración correspondiente.
* **Technician-healthTestPassed:** En este caso en concreto, nuestro cliente habla de que los test de salud anuales están "passed" o "failed".
  + **Boolean:** Equiparamos al concepto de "true" a "passed" y el de "false" a "failed";
  + **Enumerado:** Podemos recoger exactamente ese vocabulario del cliente.

En este caso particular en que tenemos una asociación tan clara entre los valores booleanos y el concepto que maneja el cliente, personalmente, lo modelaría como un booleano; fíjese que si llama Ud. al atributo "heathTestPassed" y lo modela como booleano tanto el valor true como false se interpretan perfectamente. Pero si Ud. lo modela como un atributo "healthTestStatus" de un tipo enumerado con los valores PASSED y FAILED, también es un modelo perfecto.

**-S05-D02-04:**

* **MaintenanceRecord-moment:** Es el momento en que el técnico correspondiente registra el momento de creación (CREATE), de actualización (UPDATE) o de publicación (PUBLISH) de un registro de mantenimiento; es decir, se actualiza cada vez que un técnico edita un registro de mantenimiento. Por lo tanto debe estar en el pasado o en el presente, lo que puede Ud. validar con una restricción del tipo **@ValidMoment(past=true).**
* **Aclaración del cliente:** Tanto “maintenance” como “inspection” hacen referencia a operaciones que los técnicos realizan sobre los aviones para garantizar su correcto funcionamiento, ya sea comprobaciones para certificar que todo está bien (**inspections**); o hacer reparaciones (**maintenance**). El cliente también aclara que al hablar de “**próxima fecha de inspección**”, se refiere a una fecha que el propio técnico registra como apropiada para realizar el siguiente mantenimiento, aunque esto se toma simplemente como una sugerencia por parte del técnico, no como una obligación **(OPTIONAL).** La próxima fecha de inspección será **posterior al atributo “moment"** que indica el último momento de actualización. En principio, la fecha se puede fijar en cualquier momento, pero, como indica el requisito funcional asociado, siempre y cuando un registro de mantenimiento no esté ya publicado.
* **MaintenanceRecord-nextInspection:** 
  + **A1:** No crear validación custom y permitir que haya "siguientes inspecciones" con fecha anterior a la fecha de inicio de mantenimientos actuales:
    - **Ventajas:** No se observan ventajas.
    - **Desventajas:** se permite la inclusión de datos inconsistentes (futuras inspecciones en fechas anteriores al comienzo del mantenimineto actual).
  + **A2:** Crear validación custom y obligar a que la siguiente fecha de inspección sea al menos 1 minuto posterior a la fecha de comienzo del mantenimiento actual.
    - **Ventajas:** se elimina esta inconsistencia en los datos.
    - **Desventajas:** No se observan desventajas.

Tras discutir en el foro las distintas alternativas propuestas, la 2ª solución parece la más adecuada en el contexto de Acme ANS y los requisitos del estudiante S05.

**-G-D02-04 y G-D02-05:**

* **Relación Airport-Airline:** 
  + **Many-To-One (sin entidad intermedia):**
    - **Pros:** Simplifica el modelo; cada aeropuerto se asociaría directamente a una sola aerolínea.
    - **Contras:** No representaría la realidad de que un aeropuerto es usado por múltiples aerolíneas.
  + **Many-To-Many (con entidad intermedia):**
    - **Pros:** Permite vincular varios aeropuertos a varias aerolíneas; refleja mejor la gestión de vuelos y escalas.
    - **Contras:** Añade complejidad; requiere una tabla/clase intermedia para manejar las uniones.
  + Un conjunto de letras blancas en un fondo blanco

    El contenido generado por IA puede ser incorrecto.**Solución propuesta:** Implementarlo con una entidad intermedia, observar relación entre Booking y Passenger, con BookingRecord

**-S05-D02-03, S05-D02-04 y S05-D02-05:**

* **Aclaración del cliente:**

“Los técnicos (technicians) son responsables de realizar mantenimientos (maintenances). Cada vez que un técnico realiza un mantenimiento de un avión en un determinado momento, graba un registro de mantenimiento (maintenance record) indicando qué tareas (tasks) ha realizado durante el mismo. Las tareas se eligen de un catálogo que es gestionado de forma distribuida por los propios técnicos. No es necesario que el sistema arranque con un catálogo predefinido de tareas, serán los técnicos los que las vayan registrando y publicando poco a poco. El catálogo de tareas incluye todas las que los técnicos han incluido y publicado. Por supuesto, una tarea no publicada puede ser modificada o borrada por el técnico que la ha creado y una vez publicada no son posibles ni modificaciones ni borrados”.

* **Relación MaintenanceRecords-Tasks:**
  + La relación entre los registros de mantenimiento y las tareas es lo que se suele llamar un **“conglomerado débil”;** es decir, las tareas pueden existir por sí solas independientemente de que formen o no parte de algún registro de mantenimiento y la misma tarea puede formar parte de varios registros de mantenimiento.
  + Además, el cliente nos ha aclarado que un técnico puede ver el catálogo global de tareas publicadas y añadirlas libremente a sus registros de mantenimiento según vea conveniente.
  + **Carta

    El contenido generado por IA puede ser incorrecto.Problema:** Modelar una relación que \*conceptualmente\* es de tipo weak conglomerate, two-way, many-to-many. Algo como lo siguiente:
  + Una captura de pantalla de un celular con texto e imagen

    El contenido generado por IA puede ser incorrecto.**Solución propuesta:** Utilizar una entidad intermedia que nos permite hacer explícitos los enlaces que existen entre los registros de mantenimiento y sus tareas. Algo del siguiente corte:
* **Relación MaintenanceRecord-Technician:**
  + Necesitamos saber qué técnico registra cada registro de mantenimiento; sólo ese técnico podrá verlo, editarlo o borrarlo mientras no ha sido publicado.
  + **Many-To-One:** MaintenanceRecord(\*) -> Technician (1)
* **Relación Task-Technician:**
  + Necesitamos saber qué técnico registra cada tarea; sólo ese técnico podrá verlas, editarlas o borrarlas mientras no hayan sido publicadas.
  + **Many-To-One:** Task (\*)-> Technician (1)
* Texto

  El contenido generado por IA puede ser incorrecto.**Obligatoriedad de las relaciones mencionadas:**
  + Entiendo que “**Involves**” hace referencia a esa entidad intermedia que le permite implementar la relación **ManyToMany** que conceptualmente existe entre “**Maintenance Record” y “Task**”:
    - Un registro de mantenimiento puede involucrar muchas tareas y cada tarea puede estar involucrada en múltiples registros de mantenimiento.
    - “**Involves**” es una entidad intermedia que permite almacenar enlaces entre tareas y registros de mantenimiento, por lo que las dos relaciones que tiene tiene **“optional = false”.**

**-S05-D02-03 y G-D02-06:**

* **Relación Technician-Aircraft:**
  + Las relaciones que tenemos que modelar son las **persistentes**, es decir, aquéllas que guardan algún dato sobre la relación entre dos (o más) entidades. Las relaciones causales son relaciones que se establecen entre las entidades de forma circunstancial y, en caso de modelarlas, suelen dar lugar a relaciones redundantes en la BD.
  + Diagrama, Esquemático

    El contenido generado por IA puede ser incorrecto.A partir de mi análisis de los requisitos, el modelo con el que concluyo es el siguiente: