Documentación de Análisis y Diseño de la gestión de tareas

Documentación de Análisis y Diseño de Gestión de Tareas
PMPV

Introducción

Propósito

El objetivo de esta documentación es describir el análisis y diseño de la gestión de tareas de mantenimiento tanto del fabricante como de las creadas por el ingeniero en el sistema Planificación Mantenimiento Preventivo Vehicular (PMPV).

Alcance

La documentación aborda el análisis y diseño del servicio relativo a usuarios:

- Gestión de tareas manuales.
- Gestión de tareas del fabricante.
- Asignación y eliminación de documentación a una tarea de mantenimiento.
- Asignación y eliminación de la efectividad de una tarea de mantenimiento.
- Asignación y eliminación del intervalo de una tarea de mantenimiento.
- Asignación y eliminación de materiales requeridos a una tarea de mantenimiento.

Contexto del Proyecto

Descripción del Proyecto

El sistema de planificación de mantenimiento preventivo vehicular tiene como objetivo principal gestionar el mantenimiento programado de una flota de vehículos. La aplicación facilitará la gestión de las tareas de mantenimiento proporcionadas por el fabricante como de aquellas generadas por el propio operador. Además, se gestionarán las revisiones de las tareas, notificando cualquier cambio a los ingenieros responsables de la gestión y planificación del mantenimiento de vehículos.

Los ingenieros crearán planes de mantenimiento para vehículos similares que al aplicarlos generarán, automáticamente, órdenes de trabajo. Estas, serán gestionadas por los mecánicos, los cuales también podrán crear nuevos trabajos debido a la detección de errores durante la realización de sus trabajos. Además, la aplicación gestionará los diferentes tipos de usuario (ingenieros, mecánicos, administradores, etc..) que podrán acceder de forma segura a las diferentes funcionalidades de esta.

Cabe destacar que el plan de mantenimiento preventivo contribuye a prevenir fallos y garantizar el correcto funcionamiento de los vehículos. De hecho, la aplicación se deberá diseñar para que gestione un crecimiento y sea escalable y esté en la nube, con procesos automatizados para minimizar errores y garantizar su calidad.

Stakeholders

Ingenieros

Requisitos

Requisitos funcionales

PMPV-4: US-2023-Como ingeniero quiero crear una tarea de mantenimiento manualmente. FINALIZADA

PMPV-5: US-2023-Como ingeniero quiero actualizar la información de una tarea de mantenimiento. FINALIZADA

PMPV-11: US-2023-Como ingeniero quiero eliminar una tarea de mantenimiento. FINALIZADA

PMPV-12: US-2023-Como ingeniero quiero ver la lista de tareas de mantenimiento FINALIZADA

PMPV-22: US-2023-Como ingeniero quiero añadir un material necesario para realizar la tarea de mantenimiento FINALIZADA

PMPV-23: US-2023-Como ingeniero quiero añadir un documento a una tarea de mantenimiento FINALIZADA

PMPV-24: US-2023-Como ingeniero quiero quitar un documento a una tarea de mantenimiento. FINALIZADA

PMPV-25: US-2023-Como ingeniero quiero que las tareas de mantenimiento del fabricante (Toyota) se creen automáticamente en el sistema. FINALIZADA

PMPV-26: US-2023-Como ingeniero quiero que las tareas de mantenimiento del fabricante (Toyota) se actualicen automáticamente en el sistema. FINALIZADA

PMPV-53: US-2023-Como ingeniero quiero poder añadir la efectividad de la tarea de mantenimiento FINALIZADA

PMPV-54: US-2023-Como ingeniero quiero quitar un material necesario para realizar la tarea de mantenimiento FINALIZADA

PMPV-111: US-2023-Como ingeniero quiero que las tareas de mantenimiento del fabricante (Toyota) se eliminen automáticamente en el sistema. FINALIZADA

PMPV-113: US-2023-Como ingeniero quiero poder quitar efectividad a la tarea de mantenimiento FINALIZADA

Requisitos no funcionales

PMPV-100	RS-Usabilidad
PMPV-99	RS-Diseño Responsivo

Análisis

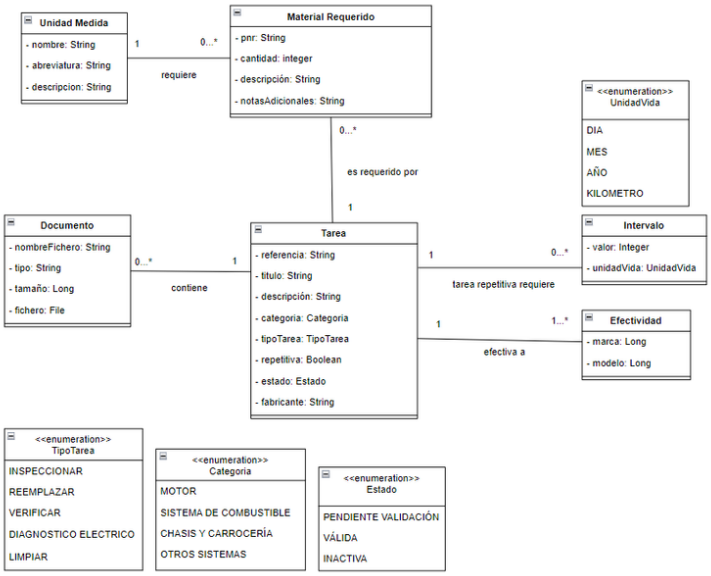
Modelo de Dominio y Restricciones de Integridad

Una tarea de mantenimiento vehicular engloba una serie de actividades que se realizan para garantizar la seguridad y el buen funcionamiento del vehículo. Estas tareas pueden ser repetitivas, en cuyo caso, es obligatorio especificar la frecuencia con la que se ejecuta. Por ejemplo, 1 vez al año o a los 15.000 kilómetros. Tal y como se puede observar en el ejemplo, se podrá introducir uno o varios intervalos.

También, es muy importante informar la efectividad de la tarea, es decir, a qué marca o modelo de vehículo aplicará. Esto garantiza que, al aplicar un plan de mantenimiento a uno o varios automóviles, solo se generen los trabajos que realmente correspondan. Un dato a tener en cuenta es que al menos se deberá informar la marca y, en caso de que aplique a todas, se podrá especificar esto fácilmente.

En caso de que las tareas requieran algún material especial, el usuario podrá asociarlo indicando el artículo, la cantidad necesaria y la unidad de medida, tales como kilogramos, unidades de ese producto o litros, entre otros. Del mismo modo, podrá adjuntar documentación técnica de la tarea, como los pasos necesarios que hay que realizar para llevar a cabo el trabajo.

Una vez que los requisitos han quedado claros, se ha procedido a realizar el modelo de dominio relativo al servicio de la gestión de tareas y a especificar las restricciones de integridad. A continuación, se presenta el modelo de dominio donde se puede observar el patrón de análisis de cantidad que facilitará la gestión de las diferentes medidas de una manera fácil.



Modelo de dominio del Servicio Tarea

Cabe destacar que la efectividad contendrá la referencia de la marca y el modelo ya que estas entidades serán gestionadas en otro servicio. Además, se ha querido enfatizar la importancia de la efectividad ya que no solo incluirá la marca y el modelo, sino que se irá ampliando e incluirá otro tipo de condiciones.

Restricciones de integridad:

Estas reglas que aplican al diseño de la base de datos proporcionan las normas que deben cumplir los datos para garantizar la consistencia, la coherencia y la fiabilidad de la información que se almacena en el sistema.

- Intervalos y Unidades:
- Solo una tarea repetitiva puede tener frecuencia.

Una tarea de mantenimiento puede tener varios intervalos, pero nunca debe tener el mismo intervalo (valor + unidad) o unidad de vida (unidad).

El valor del intervalo deberá ser superior a 0.

- Documentos Técnicos:
- Una tarea puede tener varios documentos técnicos, pero no puede tener asociado un documento con el mismo nombre.
- El tamaño del documento debe ser superior a 0.

- Materiales Requeridos:
- El valor de la cantidad del material requerido debe ser superior a 0.

Una tarea puede tener diferentes materiales requeridos, pero no puede tener la misma referencia de material (PNR) con la misma unidad de medida, es decir, es posible tener aceite en litros y garrafas, pero no dos veces lo mismo (PNR + unidad).

- Identificación Tareas:
- Una tarea se identifica por el campo referencia. Por lo tanto, no pueden existir dos tareas con la misma referencia.

Operaciones del Sistema

Del mismo modo que se realizó en el Sprint anterior se han identificado los comandos y consultas que se corresponden con los endpoints del Servicio Tarea. A continuación, se presentan las correspondientes listas:

Comandos:

Actor	Caso de uso	Nombre comando	Descripción
Ingeniero	Crear tarea manual	*crearTareaManual()*	Crear tarea manual desde la aplicación
Ingeniero	Modificar tarea manual	*modificarTareaManual()*	Modificar tarea desde la aplicación.
Ingeniero	Eliminar tarea manual	*elminarTareaManual()*	Eliminar tarea desde la aplicación.
Fabricante	Crear tarea fabricante	*crearTareaFabricante()*	La tarea es creada desde el exterior por un fabricante
Fabricante	Modificar tarea fabricante	*modificarTareaFabricante ()*	La tarea es modificada desde el exterior por un fabricante
Fabricante	Eliminar tarea fabricante	*elminarTareaFabricante()*	La tarea es eliminada desde el exterior por un fabricante
Ingeniero/Fabricante	Añadir documento a tarea	*anadirDocTarea()*	Se añade un documento a la tarea.
Ingeniero/Fabricante	Quitar documento a tarea	*eliminarDocTarea()*	Se elimina un documento de la tarea.
Ingeniero/Fabricante	Añadir material requerido a tarea	*anadirMaterialTarea()*	Se añade un material requerido a la tarea.
Ingeniero/Fabricante	Modificar material requerido de tarea	*modificarMaterialTarea()*	Se modifica un material requerido a la tarea.
Ingeniero/Fabricante	Quitar material requerido a tarea	*eliminarMaterialTarea()*	Se elimina un material requerido de la tarea.
Ingeniero/Fabricante	Añadir intervalo a tarea	*anadirIntervalo()*	Se añade un intervalo a la tarea.
Ingeniero/Fabricante	Modificar intervalo de tarea	*modificarIntervalo()*	Se modifica un intervalo a la tarea.
Ingeniero/Fabricante	Quitar intervalo a tarea	*eliminarIntervalo()*	Se elimina un intervalo de la tarea.
Ingeniero/Fabricante	Añadir efectividad a tarea	*anadirEfectividadTarea()*	Se añade una efectividad a la tarea.
Ingeniero/Fabricante	Quitar efectividad a tarea	*eliminarEfectividadTarea()*	Se elimina una efectividad de la tarea.

Comandos Gestión de Tareas

Consultas:

Actor	Caso de uso	Nombre comando	Descripción
Ingeniero	Ver Lista Tareas	*consultarTareas()*	Recupera la información de las todas las tareas.

Consultas Gestión de Tareas

Finalmente, teniendo en cuenta los casos de uso identificados anteriormente, se presenta el diagrama de casos de uso que muestra la iteración que tiene el ingeniero y el fabricante con el sistema:

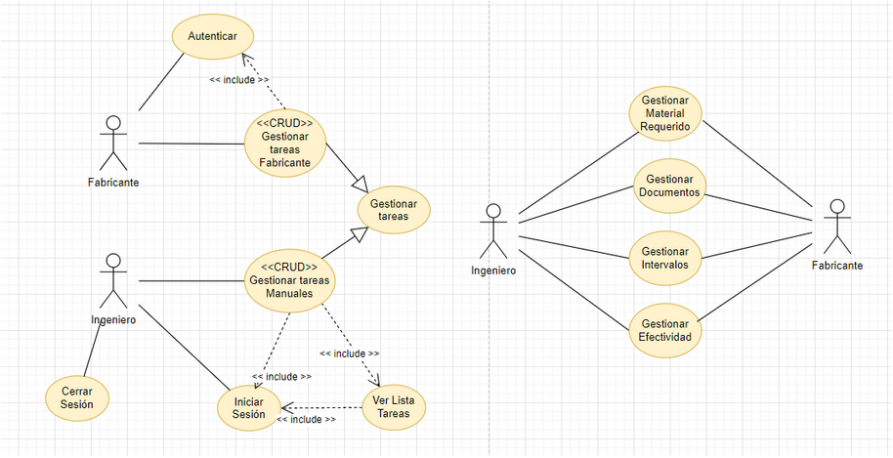


Diagrama de casos de uso de la gestión de tareas

En el diagrama, se puede observar cómo el usuario puede crear, modificar y eliminar tareas manuales, pero ello requerirá el inicio de sesión de la aplicación. Sin embargo, el fabricante al hacerlo desde el exterior deberá autenticar el proceso de intercambio de información.

Diseño

Base de Datos

Dado que se aplica el Patrón base de datos por servicio, donde cada servicio tiene su propia base de datos, surge la necesidad de gestionar transacciones distribuidas de manera consistente y tolerante a fallos. Para abordarla, se aplica el Patrón CQRS, visto anteriormente, junto con el Patrón Saga que consiste en dividir el proceso transaccional en pasos más pequeños, permitiendo, en caso de fallo, revertir las acciones realizadas en los pasos anteriores.

Tal y como se realizó en el *Sprint* anterior, se ha llevado a cabo el diagrama relacional que muestra claramente las relaciones entre las diferentes entidades que forman parte de la gestión de tareas. Además, se han tenido en cuenta las restricciones de integridad indicadas anteriormente. A continuación, se exponen algunas de las relaciones que se han establecido:

La tabla “TAREA_MATERIAL_REQUERIDO” tiene una relación con la tabla “UNIDAD_MEDIDA” a través de la clave foránea “ID_UNIDAD_MEDIDA”, que referencia al campo “ID_UNIDAD_MEDIDA” en la tabla “UNIDAD_MEDIDA”. Esta relación permite asociar la unidad de medida al material de una tarea.

La tabla “TAREA_MATERIAL_REQUERIDO” tiene una relación con la tabla “TAREA” a través de la clave foránea “ID_TAREA”, que referencia al campo “ID_TAREA” en la tabla “TAREA”. Esta relación permite asociar materiales requeridos a una tarea específica.

La tabla “TAREA_DOCUMENTO” tiene una relación con la tabla “TAREA” a través de la clave foránea “ID_TAREA”, que referencia al campo “ID_TAREA” en la tabla “TAREA”. Esta relación permite asociar documentos a una tarea específica.

La tabla “TAREA_INTERVALO” tiene una relación con la tabla “TAREA” a través de la clave foránea “ID_TAREA”, que referencia al campo “ID_TAREA” en la tabla “TAREA”. Esta relación permite asociar intervalos a una tarea específica.

La tabla “TAREA_EFECTIVIDAD” tiene una relación con la tabla “TAREA” a través de la clave foránea “ID_TAREA”, que referencia al campo “ID_TAREA” en la tabla “TAREA”. Esta relación permite asociar marcas y modelos a una tarea específica.

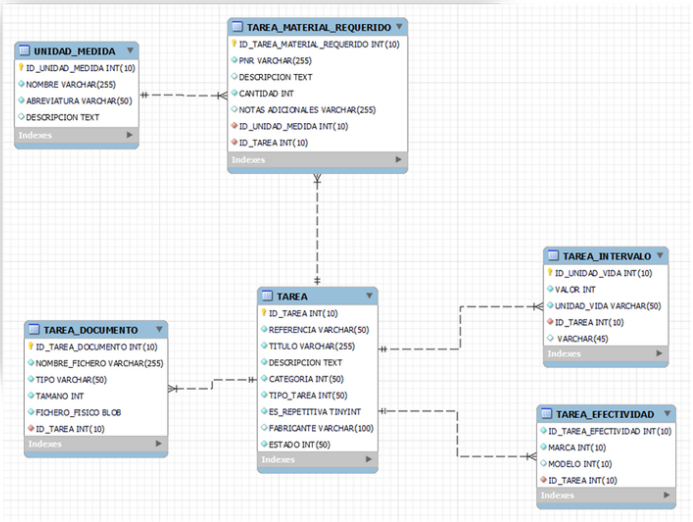


Diagrama relacional del modelo de datos de la gestión de tareas

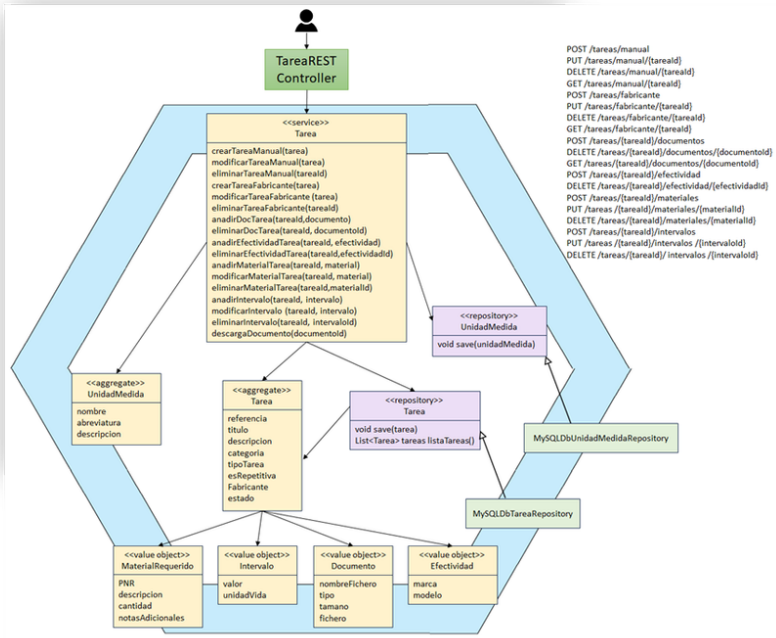
Por último, hay que indicar que, al aplicar las restricciones de integridad, el campo de la tabla “TAREA” es identificado como Unique, es decir, no se puede repetir. Del mismo modo, se ha realizado para el resto de las entidades. De modo que, en el caso de la tabla “TAREA_MATERIAL_REQUERIDO” se han considerado los tres campos “PNR”, “ID_UNIDAD_MEDIDO” y “ID_TAREA”.

Diseño del servicio

En este Sprint, se muestra el detalle del diseño del servicio de tareas, teniendo en cuenta que se aplica una arquitectura hexagonal y que en el diagrama se representan los protocolos de comunicación con el servicio, los métodos y los atributos de las diferentes clases del servicio.

En el siguiente diagrama se puede observar cómo se relacionan los componentes, como la capa de servicio interactúa con la capa de dominio y cómo los repositorios están conectados con las entidades. También, como la capa de servicios, *TareaRest Controller* y Tarea, son los responsables de manejar las solicitudes HTTP y dirigirlas a las operaciones de la capa de dominio.

Luego, en la capa de dominio se pueden ver las entidades agregadas, los objetos de valor y los atributos más destacados. Donde los agregados son conjuntos de entidades que son coherentes y los objetos de valor se usan en el contexto únicamente de la tarea.



Diseño del Servicio Tarea

Los endpoints definidos en este servicio son los siguientes:

v Para las tareas manuales:

Método: POST URL: **tareas/manual** Método: PUT URL: **tareas/manual/{tareaId}**	Método: DELETE URL: **tareas/manual/{tareaId}** Método: GET URL: **tareas/manual/{tareaId}**

v Para las tareas del fabricante:

Método: POST URL: **tareas/fabricante** Método: PUT URL: **tareas/fabricante/{tareaId}**	Método: DELETE URL: **tareas/ fabricante/{tareaId}** Método: GET URL: **tareas/fabricante/{tareaId}**

v Para los documentos de las tareas:

Método: POST

URL: **tareas/{tareaId}/documentos**

Método: DELETE

URL: **tareas/{tareaId}/documentos/{documentoId}**

Método: GET

URL: **tareas/{tareaId}/documentos/{documentoId}**

v Para las efectividades de la tarea:

Método: POST

URL: **tareas/{tareaId}/efectividad/**

Método: DELETE

URL: **tareas/{tareaId}/efectividad/{efectividadId}**

v Para los intervalos de la tarea:

Método: POST

URL: **tareas/{tareaId}/intervalos**

Método: PUT

URL: **tareas /{tareaId}/intervalos/{intervaloId}**

Método: DELETE

URL: **tareas /{tareaId}/intervalos/{intervaloId}**

v Para los materiales requeridos de la tarea:

Método: POST

URL: **tareas/{tareaId}/materiales**

Método: PUT

URL: **tareas /{tareaId}/materiales/{materialId}**

Método: DELETE

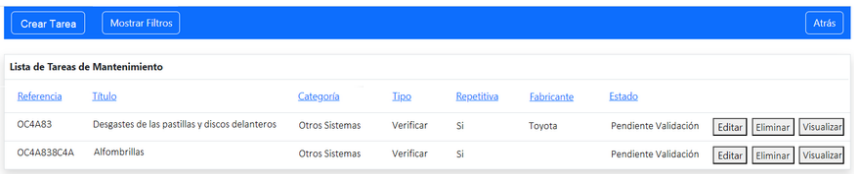
URL: **tareas /{tareaId}/materiales/{materialId}**

Interfaz de usuario

Una vez que el usuario ha iniciado sesión en el sistema, se muestra la interfaz específica del rol que tiene ese usuario. En el caso de que sea ingeniero, se presenta el siguiente menú principal desde el cual se puede acceder a la gestión de tareas.

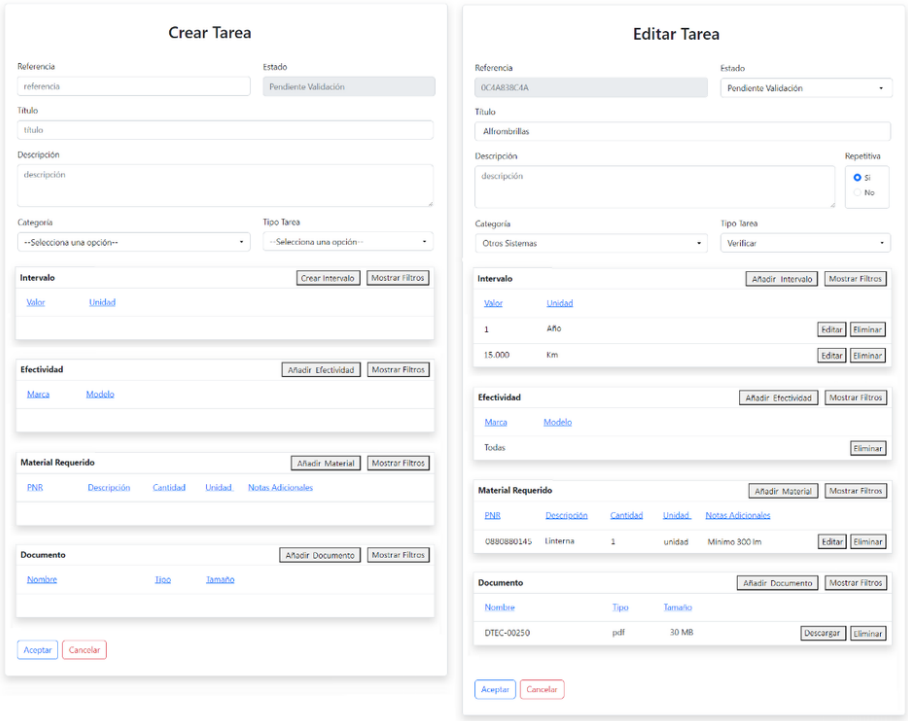
Menú Principal Rol Ingeniero

La ventana “Lista de Tareas de Mantenimiento” permite gestionar las tareas de mantenimiento manuales. Además, se visualizan todas las tareas que se han dado de alta en el sistema.



Ventana “Lista de Tareas de Mantenimiento”

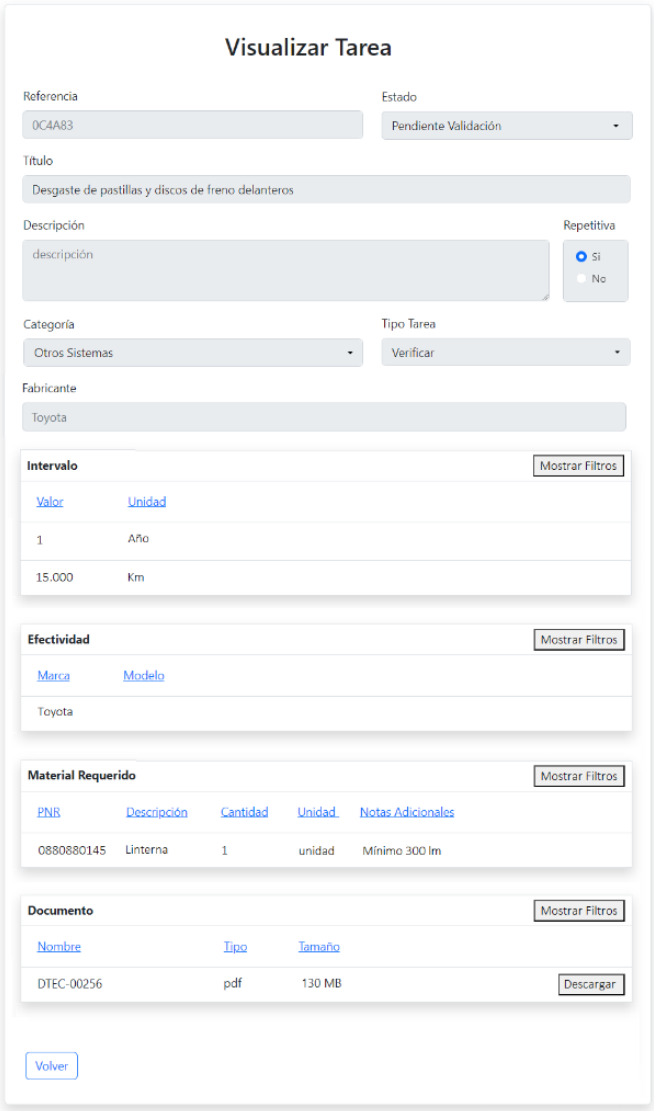
Las siguientes ventanas permiten crear y actualizar tareas manuales.



Ventana “Crear Tarea” y “Editar Tarea”

En la ventana “Crear Tarea” se puede especificar todo el detalle de la tarea, como el intervalo si es repetitiva, la efectividad, los materiales requeridos y los documentos asociados. Además, el estado se establece por defecto como pendiente de validación, y este no podrá ser modificado por el usuario. Sin embargo, en la ventana de actualización, el estado podrá ser actualizado, pero no la referencia porque es lo que identifica la tarea.

Finalmente, la ventana “Visualizar Tarea” presenta una vista de solo lectura con todo el detalle de la tarea, incluido el fabricante que la generó, si ese fuera su origen.



Ventana “Visualizar Tarea”

Automatización de pruebas

La automatización de pruebas es fundamental para verificar y validar los requisitos. Además, garantiza su cumplimiento, optimiza el proceso al integrarse con *Jenkins* y facilita la detección de fallos.

Dado que se ha aplicado el patrón de microservicios, donde varios servicios interactúan entre sí, es importante verificar que se comportan correctamente. Por ese motivo, se aplica el patrón prueba de componente de servicio.

El patrón indicado anteriormente permite probar un servicio de forma aislada a través de diferentes tecnologías como puede ser *Junits*, *ArchUnit*, *Concordium* y *Selenium*.

Sin embargo, el patrón prueba de contrato de integración de servicios permite comprobar que las APIs que un servicio proporciona a otros cumplen con las expectativas del consumidor.

Objetivo

El objetivo de la automatización del servicio tarea es:

- Validar las siguientes funcionalidades:
- Crear, actualizar y eliminar tareas manuales y automáticas.
- Añadir y eliminar documentos a una tarea de mantenimiento.
- Añadir y eliminar materiales requeridos a una tarea de mantenimiento.
- Añadir y eliminar la efectividad de una tarea de mantenimiento.

- Garantizar que se cumplen las restricciones de integridad.
- Comprobar que se integran correctamente con Jenkins CI.

Tipos de pruebas:

Prueba Unitaria

Comprueba partes específicas del código utilizando *Junit*.

Prueba de gobernabilidad de la arquitectura

Permite asegurar que la implementación que se ha llevado a cabo ha seguido los principios arquitectónicos que se han definido con *ArchUnit*.

Prueba de regresión

Sirve para detectar fallos sobre funcionalidades que han sido implementadas y que pueden verse impactadas por alguna modificación. Esto se ve claramente con los servicios, si se cambia un servicio puede afectar a su consumidor. De ahí, que se incluyan pruebas del correcto funcionamiento de las APIs. En este caso, se pueden utilizar *Junit*, *Selenium* y *Concordium*.

Prueba de aceptación:

Valida que las historias de usuario cumplen las expectativas de los *stakeholders* a través de los criterios de aceptación definidos en estas. Este tipo de pruebas se harán utilizando *Selenium* y *Concordium*.

Ejecución de pruebas:

En este punto, se define el momento en el que las pruebas son ejecutadas:

- Las pruebas unitarias se ejecutan con cada subida de código que se realiza sobre la rama.
- Las pruebas de gobernabilidad, regresión y aceptación se ejecutan automáticamente todos los días por la noche porque llevará más tiempo su ejecución.

Integración con Jenkins:

En primer lugar, se configuran los nodos de ejecución de Jenkins con las dependencias necesarias para Concordium y Selenium.

Posteriormente, se definen las etapas del pipeline incluyendo las pruebas unitarias, de gobernabilidad, de regresión y de aceptación.

Finalmente, se configura Jenkins para que se genere un informe por cada tipo de prueba con los resultados de la ejecución.

Documentación de pruebas:

Además de la automatización, es esencial proporcionar documentación que permita a los *stakeholders* identificar los escenarios que se llevarán y que se han llevado a cabo. Dado que se aplica una metodología ágil, la documentación se centrará en las pruebas de aceptación definidas en las historias de usuario y su resultado, se presentará en un formato en el que los interesados puedan comprender fácilmente, tal y como se explicará a continuación.

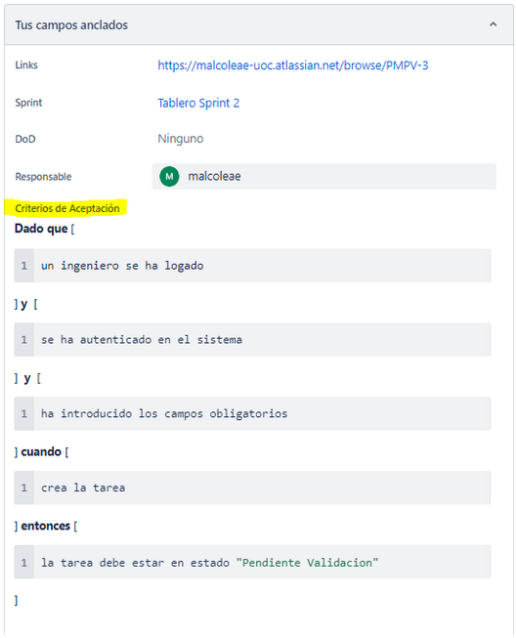
Definición de las pruebas de aceptación:

El patrón prueba de componente de servicio mencionado anteriormente, que permite reemplazar dependencias por fragmentos de código simulados, es la mejor forma de asegurar el comportamiento definido en los criterios de aceptación sobre un servicio aislado.

En este caso, las validaciones se centran en garantizar y asegurar el correcto funcionamiento de la gestión de tareas. Además, este tipo de pruebas facilita tanto la escritura como la ejecución de las mismas.

Estos criterios de aceptación están definidos en las historias de usuario creadas en *Jira* considerando aspectos como: reglas de validación, mensajes de usuario, comportamientos, características de seguridad y de incluso de rendimiento.

La estructura para definir los criterios de aceptación sigue el formato “Dado que…Cuando…Entonces” donde “**Dado que”** se corresponde con la fase de configuración de la prueba. “**Cuando”** con la fase de ejecución y “**Entonces”** con la de verificación. A continuación, se muestra uno de los escenarios que contiene la historia relativa a la creación de tareas manuales.



Criterio de aceptación relativo a la creación de una tarea

Una vez definidos los criterios de aceptación con sus correspondientes escenarios, estos son ejecutados de forma automática en Jenkins. Para que puedan ser comprendidos por los interesados no técnicos como por el equipo de desarrollo, se escriben en ***Gherkin***[1].

Gherkin permite definir escenarios que describen procedimientos mediante el desarrollo guiado por comportamiento (BDD)[2]. Además, de los elementos indicados anteriormente, ***Given-When-Then***. Por lo tanto, el escenario anterior se escribiría de la siguiente manera:

Feature: Como ingeniero, quiero crear una tarea de mantenimiento a través de la aplicación.

Scenario: Creación de una tarea manual de mantenimiento.

Given: Un ingeniero logado y autenticado y que ha introducido todos los campos obligatorios de la tarea.

When: Crea la tarea a través de la aplicación

Then: La tarea se crea con estado Pendiente Validación

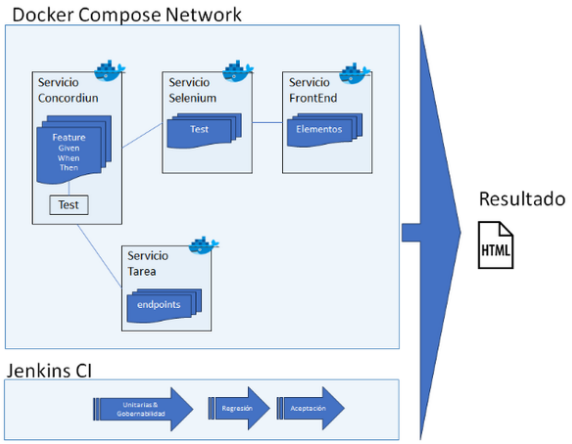
Para realizar la ejecución y visualización de los resultados, se utiliza el *framework Concordium* que facilita el desarrollo guiado por comportamiento (BDD) y presenta los resultados de manera amigable en formato HTML.

El resultado será una página HTML que contenga todos los escenarios organizados por funcionalidades mostrándose en verde cuando el escenario o conjunto de escenarios hayan pasado satisfactoriamente, pero en rojo cuando hayan fallado. A continuación, se muestra un ejemplo:

Visualización del resultado de las pruebas de los criterios de aceptación

Además, *Concordium* necesita un *framework* de pruebas por debajo, *Selenium*, que proporciona un conjunto de herramientas para interactuar con los navegadores web de manera automatizada.

Finalmente, todo se integraría en el pipeline de *Jenkins* automatizando y coordinando la ejecución de las pruebas, así como la presentación en HTML de los resultados.



Automatización de Pruebas

[1] Gherkin es un lenguaje especifico de dominio (DSL) que en este contexto describe la estructura y contenido de una historia de usuario [35].

[2] BDD o *Behaviour Driven Development* es un conjunto de prácticas destinadas a mejorar la colaboración entre desarrolladores y partes interesadas no técnicas [36].