Universitat Oberta de Catalunya

uoc.edu

Diseño de una Aplicación de Mantenimiento Preventivo Vehicular conforme a la Norma ISO/IEC/IEEE 12207:2017 con un Nivel de Madurez 3, utilizando SCRUM.

Mercedes Alcolea Escribano
Grado de Ingeniería de Informática
Ingeniería del Software

Oriol Martí Girona
Santi Caballe Llobet

19/01/2024

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Diseño de una Aplicación de Mantenimiento Preventivo Vehicular conforme a la Norma ISO/IEC/IEEE 12207:2017 con un Nivel de Madurez 3, utilizando SCRUM.*
Nombre del autor:	*Mercedes Alcolea Escribano*
Nombre del consultor/a:	*Oriol Martí Girona*
Nombre del PRA:	*Santi Caballe Llobet*
Fecha de entrega (mm/aaaa):	01/2024
Titulación:	*Grado de Ingeniería Informática*
Área del Trabajo Final:	*Ingeniería de software*
Idioma del trabajo:	*Español*
Palabras clave	*Mantenimiento, ISO, Ágil*

Resumen del Trabajo (máximo 250 palabras): *Con la finalidad, contexto de aplicación, metodología, resultados y conclusiones del trabajo.*

Esta memoria proporciona un marco de trabajo para el desarrollo de software implementando SCRUM y cumpliendo con los requisitos de la norma ISO/IEC/IEEE 12207:2017 con un nivel de madurez 3, sin que ello afecte a la agilidad del proyecto. Este enfoque se aplica durante la fase de análisis y diseño de un sistema encargado de gestionar el mantenimiento preventivo vehicular utilizando una arquitectura de microservicios y prácticas DevOps donde se presentan casos prácticos en Jira y Confluence a lo largo de los diferentes Sprints.

En el Sprint 0 se aborda la arquitectura de la aplicación desde el punto de vista de las cuatro dimensiones identificadas en el libro "Fundamentos de la Arquitectura de Software" de Mark Richards y Neal Ford en el que se reflejan: las características de la arquitectura, las decisiones arquitectónicas, los principios de diseño y la estructura del sistema.

Posteriormente, en los Sprints restantes, se abordan los siguientes servicios y temas: el servicio de usuarios gestiona los usuarios y roles, enfocándose en la autenticación y autorización del sistema mediante Spring Security; el servicio de tareas gestiona las tareas del fabricante y de los ingenieros, destacando la automatización de pruebas; y el servicio de notificaciones coordina la colaboración entre servicios mediante Kafka y Zookeeper tratando el envío de notificaciones a los ingenieros por cambios en tareas del fabricante.

Abstract (in English, 250 words or less):

This thesis provides a software development framework that implements SCRUM and aligns with the ISO/IEC/IEEE 12207:2017 standard at maturity level 3, without compromising project agility. This approach is applied during the analysis and design phase of a system tasked with managing preventive vehicular maintenance using a microservices architecture and DevOps practices, illustrating practical cases in Jira and Confluence across different Sprints.

In Sprint 0, the architecture of the application is examined through the four dimensions identified in the book "Fundamentals of Software Architecture" by Mark Richards and Neal Ford, covering architectural characteristics, decisions, design principles, and system structure.

Subsequently, in the remaining Sprints, the following services and topics are addressed: the user service, managing users and roles, focusing on system authentication and authorization via Spring Security; the task service, managing tasks from manufacturers and engineers, with an emphasis on test automation; and the notification service, coordinating collaboration between services through Kafka and Zookeeper by sending notifications to engineers due to changes in manufacturer tasks.

Agradecimientos

Me gustaría expresar mi agradecimiento a Oriol Martí Girona por su valiosa ayuda y asesoramiento a lo largo de este arduo trabajo. También, quiero agradecer el apoyo y el ánimo de toda mi familia. En particular, agradezco el apoyo de mi marido y de mi madre, quienes han sido fundamentales para llegar donde he llegado. Sin embargo, mi agradecimiento más especial va dirigido a mis hijos, Adrián y Alba, quienes han demostrado una gran paciencia al ver a su madre dedicar incontables horas de estudio, a partir de ahora, estaré más con vosotros.

Índice

Lista de figuras

Lista de tablas

1. Introducción

1.1 Contexto y justificación del Trabajo

La metodología ágil se ha vuelto cada vez más popular y ha ganado más relevancia en el campo del desarrollo de software en las últimas décadas debido a que proporciona una estructura más flexible que permite llevar a cabo entregas en un breve plazo de tiempo. Sin embargo, uno de los principios fundamentales del Manifiesto para el Desarrollo Ágil de Software destaca el valor "Software funcionando sobre documentación extensiva" [1]. Esto ha llevado a que la documentación quede en un segundo plano y, por ende, el conocimiento se haya incorporado de forma orgánica en la práctica diaria, transmitiéndose de una persona a otra en lugar de depender exclusivamente de esa información escrita. Por lo tanto, esta falta de registro puede dar lugar a la pérdida de ciertos procesos críticos, lo que conlleva riesgos significativos para la calidad del producto. Es por ello por lo que se han llevado a cabo diversos estudios relativos al ciclo de vida del desarrollo del software que integra prácticas ágiles, estándares internacionales de gestión de proyectos y calidad de software [3]. Esto es así por las siguientes razones:

- ✓ **Cumplimiento Normativo**: Muchas empresas, sobre todo aquellas en industrias reguladas, pueden estar obligadas a cumplir con determinados estándares, pero, sin embargo, quieren adoptar un enfoque más flexible y adaptable que les permita satisfacer las necesidades cambiantes del mercado o de un cliente.

- ✓ **Eficiencia y Calidad**: Dado que la documentación requerida por los procesos establecidos por los diferentes estándares puede comprometer el dinamismo que proporciona una metodología ágil, las organizaciones están interesadas en conocer cómo pueden coexistir ambos enfoques.

Teniendo en cuenta todo lo anterior, este trabajo proporcionará casos reales durante los procesos de requisitos, análisis y diseño a lo largo del ciclo de vida del proyecto teniendo en cuenta la integración de las prácticas mencionadas anteriormente, de modo que permita resolver esa falta de documentación contribuyendo a la calidad del producto, pero sin afectar la agilidad y flexibilidad del desarrollo. Para ello, esta memoria se va a centrar en sistemas de gestión de mantenimiento computarizados (CMMS)[1] porque suelen ser aplicaciones escalables y basadas en la nube centradas sobre todo en normativas relativas a la protección de datos como la ISO/IEC/IEEE 27001. Por lo tanto, como muchas empresas buscan ese tipo de arquitecturas por las ventajas que proporciona, se llevará a cabo la implementación de una aplicación software que gestiona

[1] CMMS significa "*Computerized Maintenance Management System*" y es un software que se utiliza para planificar, programar y dar seguimiento a las actividades de mantenimiento de una organización.

el plan de mantenimiento preventivo vehicular[2] permitiendo gestionar una flota de vehículos poniendo el foco en la escalabilidad, flexibilidad, tolerancia a fallos, despliegue continuo y adaptabilidad, pero sin dejar de lado, la calidad del proceso de desarrollo del producto, es decir, el producto final podrá certificarse en la norma ISO/IEC/IEEE 12207:2017[3] con un nivel de madurez 3[4].

1.2 Objetivos del Trabajo

Teniendo en cuenta el contexto y la justificación de este trabajo, se han definido los siguientes objetivos, los cuales intentan cubrir los desafíos planteados por la integración de prácticas ágiles y estándares de calidad en el desarrollo software, especialmente en las fases de requisitos, análisis y diseño:

- ✓ Desarrollar un marco de trabajo que permita la integración efectiva de prácticas ágiles y estándares de calidad en proyectos de desarrollo software relativos a la norma ISO/IEC/IEEE 12207:2017 con un nivel de madurez 3.

- ✓ Aplicar este marco de trabajo a un proyecto real como es la gestión de mantenimiento preventivo vehicular.

- ✓ Realizar el análisis y diseño de las siguientes funcionalidades:

    - Definición de la arquitectura del producto.

    - Gestión de usuarios con diferentes roles.

    - Autenticación y autorización del sistema.

    - Gestión de tareas de mantenimiento por parte de los ingenieros.

    - Automatizar la gestión de tareas de mantenimiento a partir de los datos del fabricante.

    - Informar a los ingenieros sobre cualquier modificación en las tareas procedentes del fabricante.

[2] Un plan de mantenimiento preventivo vehicular está compuesto por un conjunto de tareas programadas que se realizan a un vehículo de forma regular para prevenir fallos y garantizar su funcionamiento.
[3] La normativa ISO/IEC/IEEE 12207:2017 proporciona procesos para definir, controlar y mejorar los procesos del ciclo de vida del software de un proyecto.
[4] Un nivel de madurez 3 representa que la empresa trabaja de la misma forma, siguiendo un proceso bien definido y estandarizado.

1.3 Enfoque y método seguido

Considerando el contexto del trabajo, es esencial seleccionar una estrategia y una metodología adecuada para el producto que se va a llevar a cabo, ya que de ello dependerá el éxito del proyecto.

Respecto a la estrategia de desarrollo de producto, se ha optado por la creación de uno nuevo, aunque actualmente existen diversas aplicaciones relativas al mantenimiento planificado de vehículos. Sin embargo, el proceso de desarrollo tendrá en cuenta las siguientes premisas:

- ✓ **Metodología ágil**. Se abordará el desarrollo de software con un enfoque flexible, colaborativo, adaptable y de entrega continua.

- ✓ **Integración continua**. Se centra en la automatización de los procesos de desarrollo de software.

- ✓ **Calidad en el proceso del ciclo de vida del software** centrándose en las fases de requisitos, análisis y diseño.

Para garantizar la integración de los estándares con las prácticas ágiles desde el inicio del proceso, abarcando las fases de requisitos, análisis y diseño, se ha optado por comenzar desde cero. Esto proporciona una visión completa de los procedimientos y facilita su clarificación.

Una vez está claro el enfoque del producto, el foco se ha puesto en las metodologías ágiles. Dado que se ofrecen distintas soluciones como marco de trabajo, se ha optado por analizar Scrum y Kanban atendiendo el tipo de proyecto que se va a llevar cabo. A continuación, se indica una breve descripción de cada una de ellas:

- ✓ **Scrum** es un enfoque ágil para el desarrollo de servicios y productos que se centra en roles específicos, actividades y artefactos.

- ✓ **Kanban**: es otro enfoque ágil que se centra en representar el estado del proceso en un tablero visual y no tiene un tiempo predeterminado para realizar una serie de tareas.

Dado que existe la necesidad de realizar entregas continuas, regulares y predecibles a lo largo de la ejecución del trabajo, se ha optado por aplicar Scrum**.** El motivo por el que se ha seleccionado este enfoque es porque permite tener un mayor control y seguimiento de las tareas que se van realizando durante determinados periodos de tiempo. Pero, para implementar Scrum de manera efectiva, es esencial comprender y aplicar los siguientes elementos que son fundamentales en esta metodología:

Roles:

- ✓ *Product Owner*: Esta es la persona que tiene un conocimiento profundo del producto y se encarga de definir y gestionar sus requisitos.

- ✓ *Scrum Master*: Esta es la persona que se encarga de que se aplique de forma correcta Scrum, además de solventar los obstáculos que se produzcan durante el proceso.

- ✓ Equipo de desarrollo: Estas son las personas que se encargan de desarrollar los requisitos requeridos de manera independiente y autoorganizada del producto.

Artefactos:

- ✓ *Product Backlog*: Lista priorizada de requisitos a realizar que agregan valor al producto.

- ✓ *Sprint Backlog*: Lista de requisitos que se desarrollarán durante un Sprint específico por parte del equipo de desarrollo.

- ✓ Incremento: Lista de funcionalidades completadas durante un Sprint y que están listas para ser entregadas.

Actividades:

- ✓ *Sprint*: Periodo de tiempo finito y corto en el que se lleva a cabo el desarrollo.

- ✓ *Sprint Planning*: Reunión en la que el equipo de Scrum planifica el trabajo y lo incluye en el *backlog* del Sprint.

- ✓ *Daily Scrum*: Reunión diaria de seguimiento del Sprint.

- ✓ *Sprint Execution*: Ejecución del Sprint en el que se llevan a cabo las tareas planificadas que están incluidas en el Sprint Backlog.

- ✓ *Sprint Review*: Reunión en la que se presenta el desarrollo que se ha llevado a cabo a los interesados del producto.

- ✓ *Sprint Retrospective*: Reunión final del Sprint en la que se hace una valoración del trabajo realizado.

- ✓ *Product Backlog Grooming*: Revisión y refino del *Product Backlog*.

Por último, se tendrá en consideración las prácticas DevOps que permite unificar el desarrollo y las operaciones mediante la automatización de los procesos que veremos más en profundidad en la planificación del producto.

1.4 Planificación del Trabajo

Teniendo en cuenta que la fecha de inicio del trabajo de fin de grado es el 27 de septiembre de 2023 y su entrega se debe realizar el 19 de enero de 2024, se ha estimado un esfuerzo de 3 horas diarias, es decir, se invertirá un total de 339 horas en realizar la memoria y la presentación del TFG. Por lo tanto, el esfuerzo que se estima por tarea es el siguiente:

	Inicio	Fin	Días	Esfuerzo (horas)
PAC1	27/09/2023	11/10/2023	14	42
Investigación	27/09/2023	07/10/2023	10	30
Documentación	07/10/2023	11/10/2023	4	12
PAC2	12/10/2023	23/11/2023	42	126
Planificación Producto	12/10/2023	01/11/2023	20	60
Sprint 0	01/11/2023	23/11/2023	22	66
PAC3	23/11/2023	02/01/2024	40	120
Sprint 1	23/11/2023	07/12/2023	14	42
Sprint 2	07/12/2023	21/12/2023	14	42
Sprint 3	21/12/2023	02/01/2024	12	36
Memoria y presentación	02/01/2024	19/01/2024	17	51
Documentación	02/01/2024	19/01/2024	17	51
		Total		339

Figura 1 Estimación Esfuerzo Tareas

Por último, se muestra la planificación temporal de cada tarea utilizando un diagrama de Gantt:

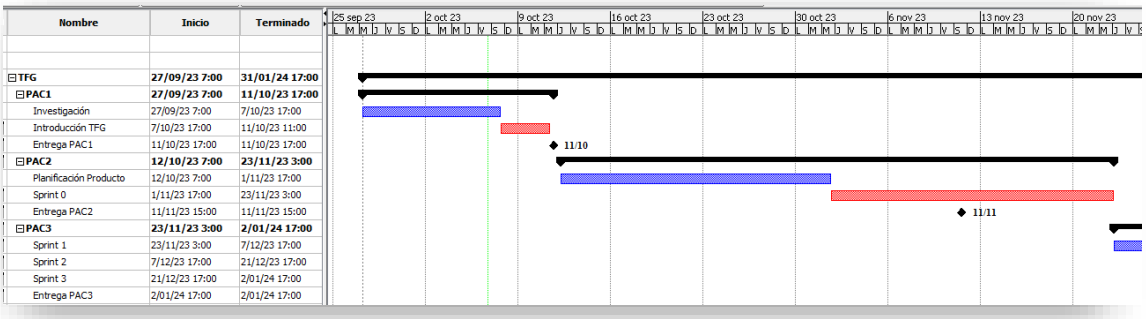

Figura 2 Planificación TFG

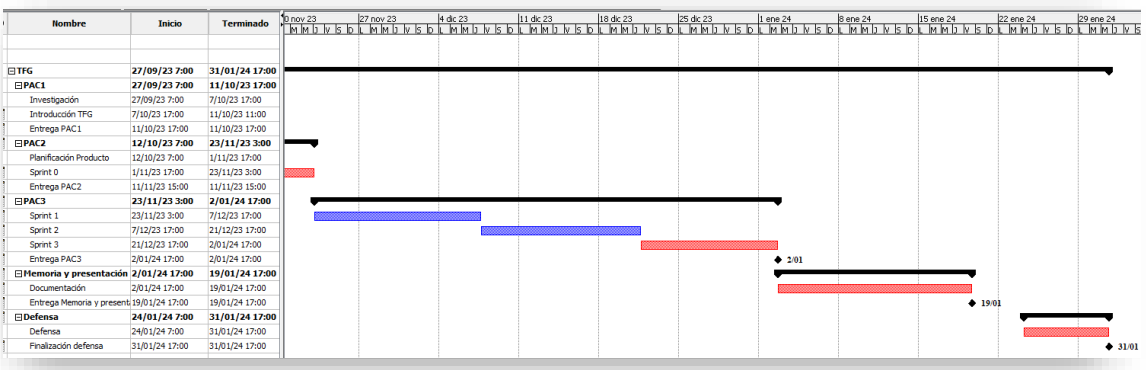

Figura 3 Planificación TFG

1.5 Breve sumario de productos obtenidos

El objetivo de este apartado es dar una visión general de los logros que se han obtenido a lo largo de los siguientes capítulos. Por lo tanto, a continuación, se presenta una lista que los enumera:

- ✓ Diseño de una aplicación de software, aplicando los principios de metodologías ágiles y estándares de calidad según la normativa ISO/IEC/IEEE 12207:2017 con un nivel de madurez 3, diseñada para la gestión de mantenimiento preventivo vehicular desde la obtención de los requisitos hasta la fase de análisis y diseño.

- ✓ Un marco de trabajo que ilustra cómo integrar eficazmente prácticas ágiles y estándares de calidad en proyectos de desarrollo de software.

- ✓ Análisis y diseño de las siguientes funcionalidades: gestión de usuarios, autorización y autenticación en el sistema, gestión de las tareas de mantenimiento por parte del ingeniero, automatización de la generación de las tareas de mantenimiento del fabricante y de la revisión de estas con la correspondiente notificación a los responsables de ingeniería.

- ✓ Resultados basados en la aplicación práctica de estos enfoques en un proyecto real en las *Review* de los diferentes *Sprints*.

1.6 Breve descripción de los otros capítulos de la memoria

A continuación, se expone una breve descripción de los capítulos que contiene la memoria y que permitirá tener una visión global del trabajo:

Capítulo 2 se centra en la planificación del producto, es decir, se define la forma de trabajar y se lleva a cabo la gestión y priorización de los requisitos.

Capítulo 3, sprint 0, se establecen los pilares de la arquitectura a través de la realización del análisis y diseño de esta.

Capítulo 4, sprint 1, se lleva a cabo la gestión de usuarios y autentificación.

Capítulo 5, sprint 2, se lleva a cabo la gestión e integración de las tareas de mantenimiento en el sistema.

Capítulo 6, sprint 3, se gestiona las revisiones de las tareas con la correspondiente notificación a los ingenieros.

Conclusiones detalla los desafíos que se han encontrado, una reflexión de los objetivos logrados y un análisis crítico del seguimiento y planificación de la metodología a lo largo del producto.

Adicionalmente, los capítulos relativos a la ejecución de un Sprint se han estructurado siguiendo los principios de Scrum para que se vea de una forma clara su aplicación.

2. Planificación inicial del producto

2.1 Consideraciones generales.

Para desarrollar la aplicación de manera efectiva y garantizar el éxito del proyecto, es fundamental comprender el producto que se tiene que implementar. Esto implica conocer muy bien las necesidades y restricciones de los diferentes *stakeholders*, ya que a partir de esta información se obtendrán los requisitos que definirán el comportamiento esperado de la aplicación y las limitaciones de las operaciones del sistema.

Por lo tanto, uno de los puntos que tratará este capítulo, son las actividades de ingeniería de requisitos: obtención de requisitos, gestión de requisitos, documentación de requisitos, validación de requisitos y verificación de requisitos. Sin embargo, los dos últimos procesos se centrarán en su definición. A continuación, se expone una breve descripción de cada uno de ellos:

- ✓ **Obtención de requisitos**: consiste en recopilar la lista de objetivos que los *stakeholders* quieren que el sistema satisfaga.

- ✓ **Gestión de requisitos**: consiste en la obtención, priorización, revisión y seguimiento de los requisitos candidatos.

- ✓ **Documentación de requisitos**: consiste en guardar un registro de los requisitos candidatos siguiendo las buenas prácticas porque será la base del desarrollo.

- ✓ **Validación de requisitos**: asegura que los requisitos cubren las necesidades y expectativas de los *stakeholders*.

- ✓ **Verificación de requisitos**: asegura que el software se haya implementado teniendo en cuenta la especificación de los requisitos.

Otros de los puntos que se tratará, es la calidad de los procesos. Dado que los requisitos son la base del desarrollo del software, es fundamental mejorar el proceso y su calidad. Para lograr el primer objetivo, mejorar el proceso, es esencial seguir las buenas prácticas definidas por el modelo **CMMI-DEV v2.0**[5], que permite alcanzar un nivel de madurez 3. Este nivel de madurez es en el que está englobado el área del desarrollo de requisitos y permite que los procesos, método y herramientas estén bien definidos.

[5] *Capability Madurity Mode Integration (CMMI)* es un modelo desarrollado por *Software Engineering Institute* para la mejora de los procesos de las empresas de software y *CMMI-DEV* se centra en los del desarrollo de software.

Sin embargo, para conseguir el objetivo de la calidad, se debe cumplir la normativa **ISO/IEC/IEEE 12207:2017** ya que tener un grado de madurez 3 no garantiza la calidad del producto. Esto es así porque se ha podido realizar una definición poco clara de los requisitos.

Por lo tanto, y dado que ambos estándares solo indican qué hay que hacer o cumplir, pero no especifican cómo se tienen que aplicar, en los puntos posteriores, se proporciona un enfoque práctico y específico para garantizar la calidad en los procesos de desarrollo de software.

2.2 Cumplimiento de áreas de prácticas CMMI DEV v2.0 con SCRUM.

En este punto, se detalla cómo se podrían alinear las prácticas recomendadas por CMMI con los artefactos y prácticas de Scrum en un marco de trabajo relativo al ciclo de vida del desarrollo del software. De modo que permita mejorar la madurez de los procesos y alcanzar un nivel de madurez 3.

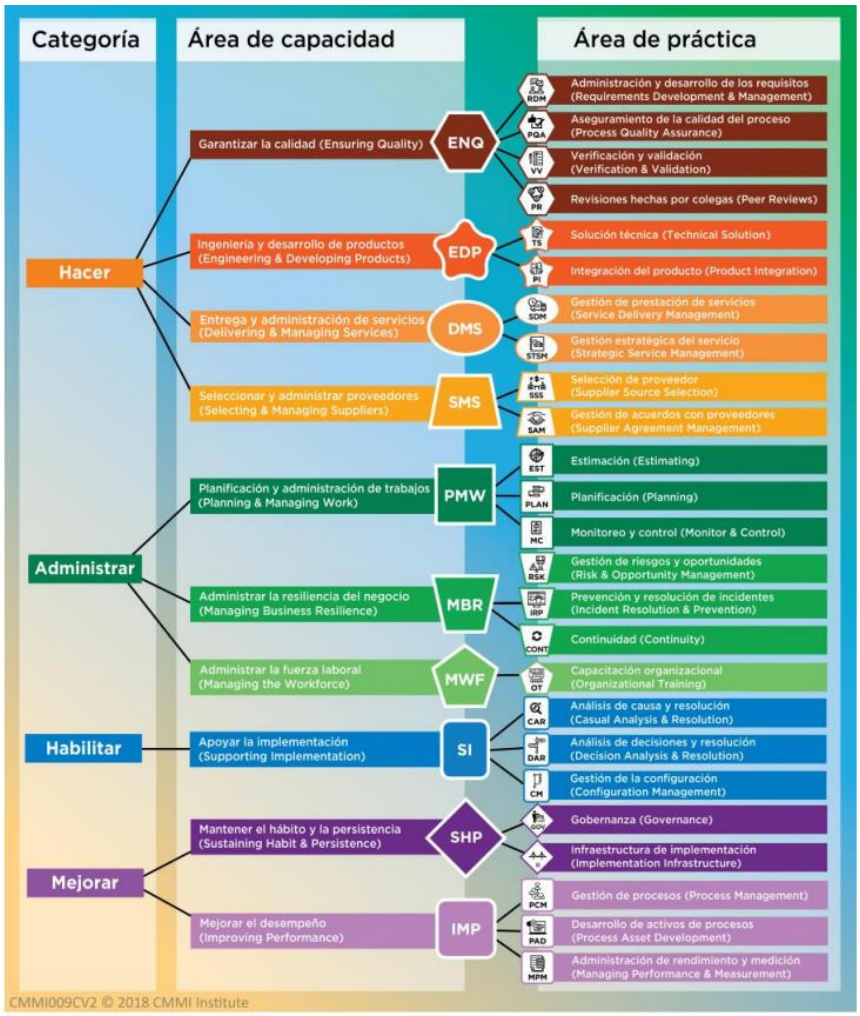

Figura 4 Categorías, áreas de capacitación y áreas de prácticas definides por CMMI [12]

Área de Práctica de CMMI-DEV v2.0	Artefactos y Prácticas de Scrum
Administración y desarrollo de los requisitos (RDM)	La gestión de requisitos es cubierta por: ➢ El *Product Backlog* que contiene los requisitos (*EPICs* y *User Stories)*. ➢ Las reuniones de refinamiento que permite revisarlos.
Aseguramiento de la calidad del proceso (PQA)	Para garantizar la calidad del proceso, se establece la definición de terminado (DoD).
Verificación y Validación (VV)	Para asegurar que el producto cumple con las expectativas de los *stakeholders*: ➢ El trabajo realizado será revisado en la *Sprint Review*. ➢ Se especificarán criterios de aceptación en las historias de usuario.
Revisiones hechas por colegas (PR)	Para asegurar la calidad del código se realizan revisiones por pares, pair programing.
Solución Técnica (TS)	Para asegurar la definición y diseño de la solución técnica, se incluirá como un requisito más de la definición del trabajo realizado (DoD). De modo que la historia de usuario contenga la documentación necesaria: ➢ Modelo de dominio. ➢ Diagrama de componentes. ➢ Diagrama de despliegue. ➢ Diagrama de secuencia. ➢ Diagrama de actividad.
Integración del producto (PI)	Para la integración del producto se aplicará la Integración Continua (CI) y el despliegue continuo (CD).
Gestión de la configuración (CM)	Para controlar los cambios de los elementos de software y de otros artefactos relacionados se llevará a cabo: ➢ Una estrategia de gestión de ramas. ➢ Un sistema de control de versiones.

Tabla 1 Cumplimiento de áreas prácticas CMMI con SCRUM

2.3 Cumplimiento de resultados ISO/IEC/IEEE 12207:2017 con SCRUM.

En este otro punto, se detalla cómo se alinean los resultados recomendados por ISO/IEC/IEEE 12207:2017 con los artefactos y prácticas de *Scrum* teniendo en cuenta el marco de trabajo indicado anteriormente. De modo que proporcione calidad a los procesos.

Definición de requisitos de las partes interesadas	
Resultado	**Artefactos y Prácticas de Scrum**
Identificación de partes interesadas.	El *Product Owner* identifica en las historias de usuario a las partes interesadas.
Se definen las características requeridas y el contexto de uso de las capacidades y los conceptos de las etapas del ciclo de vida, incluyendo conceptos operacionales.	Será especificado en las historias de usuario.
Se definen las restricciones del sistema.	Las restricciones se incluirán en el *Product Backlog* y podrán asociarse a las historias de usuario.
Las necesidades de las partes interesadas son definidas.	El *Product Owner* se encarga de reflejar las necesidades de las partes interesadas en el *Product Backlog* a través de reuniones de refinamiento.
Las necesidades de las partes interesadas son priorizadas y transformadas en requisitos claramente definidos.	El *Product Owner* prioriza las historias de usuario que reflejan con claridad las necesidades de las partes interesadas.
Se definen medidas críticas de rendimiento.	En las historias de usuario se definen esas medidas críticas a través de los criterios de aceptación.

Las partes interesadas aceptan que sus necesidades y expectativas se reflejan adecuadamente en los requisitos que se logran.	Se realizará la demostración y revisión de los requisitos realizados en el *Sprint Review* para que las partes interesadas puedan evaluar si sus necesidades han sido reflejadas adecuadamente.
Los sistemas o servicios habilitantes necesarios para las necesidades y requisitos de las partes interesadas están disponibles.	El equipo de desarrollo con la ayuda del *Product Owner* se encargan de asegurar que los sistemas o servicios estén disponibles antes de realizar la integración.
Existe trazabilidad de los requisitos de *stakeholder*, a los *stakeholders* y a sus necesidades.	Las *EPICs* definen requisitos de alto nivel y representan objetivos de los *stakeholders.* Por lo tanto, se definen las necesidades de los *stakeholders* y los requisitos que los cubrirán.

Tabla 2 Cumplimiento de resultados ISO/IEC/IEEE 12207:2017 con SCRUM

Definición de requisitos del sistema	
Resultados	**Artefactos y Prácticas de Scrum**
Se describe el sistema incluyendo interfaces, funciones y límites.	Tanto las *EPIs* como las historias de usuario recopilarán esa información.
Se describen los requisitos del sistema/software (funcionales, de rendimiento, de proceso, no funcionales y de interfaz) y las restricciones del diseño.	Las historias de usuario podrán ser funcionales, de rendimiento, de proceso, no funcionales y de interfaz. Además, se podrán crear restricciones que podrán ser asociadas a las *User Stories*.
Se definen medidas críticas de desempeño o rendimiento.	Se definen criterios de aceptación en las historias de usuario.
Se analizan los requisitos del sistema/software.	Inicialmente, el *Product Owner* analiza los requisitos que se incluirán en el *Sprint*. Posteriormente, en el *Sprint Planning,* el equipo de desarrollo los estima. Por lo tanto, si durante la estimación, se detectara cualquier nueva restricción, limitación o detalle de la funcionalidad será incluido en la historia.
Están disponibles todos los sistemas o servicios habilitantes necesarios para la definición de los requisitos del sistema/software.	Tanto en las *EPIs* como en las historias de usuario y se documentarán las dependencias con sistemas o servicios externos y/o internos.
Se desarrolla la trazabilidad de los requisitos del sistema/software a los requisitos de las partes interesadas.	Las *EPICs* definen requisitos de alto nivel y representan objetivos de los *stakeholders* y las historias de usuario siempre estarán asociadas a una *EPIC.* Automatización de los criterios de aceptación.

Tabla 3 Cumplimiento de resultados ISO/IEC/IEEE 12207:2017 con SCRUM

Definición de la arquitectura	
Resultados	**Artefactos y Prácticas de Scrum**
Los requisitos identificados de los interesados son abordados por la arquitectura.	Las características de la arquitectura se basarán en los requisitos de los interesados y se especificará en la *EPIC*.
Se desarrollan puntos de vista de la arquitectura.	Diagrama de Arquitectura.
Se definen contexto, límites e interfaces externas del sistema.	Diagrama de Contexto.
Se desarrollan vistas y modelos de arquitectura del sistema.	Diagrama de Componentes. Diagrama de Despliegue. Diagrama de Flujo de Datos.

Conceptos, propiedades, características, comportamientos, funciones o restricciones que sean significativas para las decisiones sobre arquitectura del sistema son asignadas a las entidades de la arquitectura.	En las historias de usuarios se especificará todas esas necesidades.
Se identifican los elementos y las interfaces del sistema.	En las historias de usuarios se especificará todas esas necesidades y se pueden crear etiquetas que permitan identificarlos más fácilmente.
Se evalúan las arquitecturas candidatas.	Durante el *Sprint Planning* y los refinamientos se evaluarán y si es necesario se llevará a cabo durante la ejecución de un *Sprint* con su correspondiente historia de usuario de tipo no funcional.
Se logra una base arquitectónica para los procesos a lo largo del ciclo de vida.	Habrá *EPIs* e historias de usuario que cubran esas necesidades y puedan ser planificados en *Sprints*.
Se logra la alineación entre arquitectura y requisitos y características del diseño.	Se realizará la demostración y revisión de los requisitos realizados en el *Sprint Review* para que las partes interesadas puedan evaluar si sus necesidades han sido reflejadas adecuadamente. Automatización de pruebas y monitorización del sistema.
Los sistemas habilitados o los servicios necesarios para la arquitectura estén disponibles.	Se crearán *EPIs* e historias de usuario que cubran estas necesidades y se documentarán las dependencias que existan.
Existe trazabilidad entre los elementos de arquitectura y los requisitos del sistema o SW.	Las *EPICs* definen requisitos de alto nivel y representan objetivos de los *stakeholders* y las historias de usuario siempre estarán asociadas a una *EPIC.* Automatización de los criterios de aceptación.

Tabla 4 Cumplimiento de resultados ISO/IEC/IEEE 12207:2017 con SCRUM

2.4 Selección de herramientas para la documentación de requisitos.

Una vez claro cómo *Scrum* puede ser utilizado para cumplir con los requisitos y resultados esperados por las normativas y estándares expuestas anteriormente, se procede a seleccionar las herramientas que se utilizaran para documentar los requisitos. Para ello, se opta por utilizar dos herramientas que son ampliamente utilizadas en este tipo de metodologías ágiles:

- ***Jira*** porque ofrece soporte para Scrum, permite gestionar historias de usuario y personalizar artefactos que servirán de plantillas.

- **Confluence** porque está integrado con Jira, facilita la generación de documentación y permite trazarla fácilmente con historias de usuario.

2.5 Gestión de requisitos y ciclo de vida del desarrollo.

En esta sección, se expondrán las actividades clave que conforman el ciclo de vida del desarrollo del proyecto teniendo en cuenta el marco de trabajo definido anteriormente. Además, se especifican acciones concretas en Jira relativas a la estandarización de procesos de modo que facilite la comprensión y la implementación de estos.

Revisión y refino del Product Backlog (Product Backlog Grooming):

En esta actividad, el *Product Owner* se reúne con los *stakeholders* para realizar la revisión y refinamiento del *Product Backlog*. El objetivo de esta reunión consiste en aclarar y definir las necesidades de las partes interesadas permitiendo al *Product Owner* gestionar los requisitos de una forma efectiva y que estos queden priorizados y documentados correctamente en el *Product Backlog* de Jira.

En el *Product Backlog* de Jira, se utilizan tres tipos de incidencias para documentar estas necesidades: ***EPICs***, que representan requisitos de alto nivel y que abarcan un conjunto de funcionalidades relacionadas o un objetivo, ***User Stories***, que representan requisitos específicos y que se centran en una funcionalidad o característica del sistema desde la perspectiva del usuario y ***Restricciones***, que representan limitaciones que tenga el sistema.

Estos tres elementos son configurados para establecer una plantilla que permita mejorar la gestión de requisitos, el cumplimiento con los estándares y prácticas de *CMMI* y alcanzar un nivel de madurez 3. A continuación, se expone la configuración que se ha realizado:

EPIC:

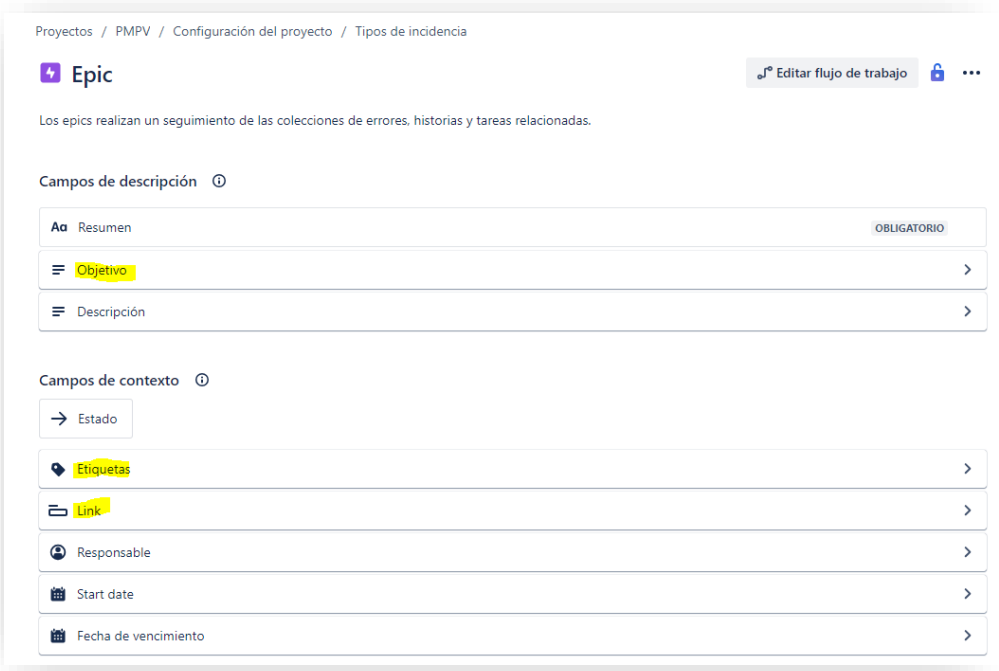

Figura 5 Plantilla EPIC en JIRA

Resumen: REQ-Nombre Identificativo Requisito Alto Nivel.

Objetivo: Se identifican los *stakeholders* y sus objetivos.

Descripción: Se exponen los requisitos necesarios para alcanzar el objetivo y las dependencias o restricciones que haya.

Etiquetas: para clasificar e identificar funcionalidades o características específicas.

Categoría. Por ejemplo, mantenimiento.

Link: Documentación relevante para el requisito, como minutas de reuniones con partes interesadas documentadas en *Confluence* o detalles adicionales relacionados con los requisitos.

USER STORY:

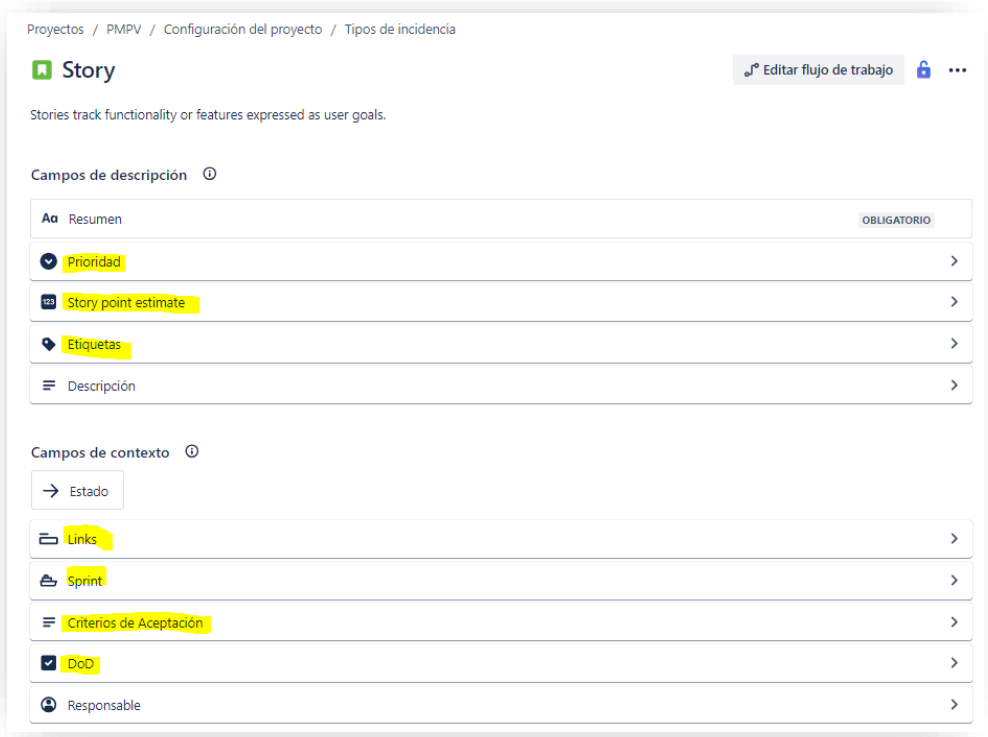

Figura 6 Plantilla User Story en JIRA

Resumen: US-Año-**Como** [perfil] **quiero** [objetivo] **para** [resultado]

Prioridad: Por defecto, medio.

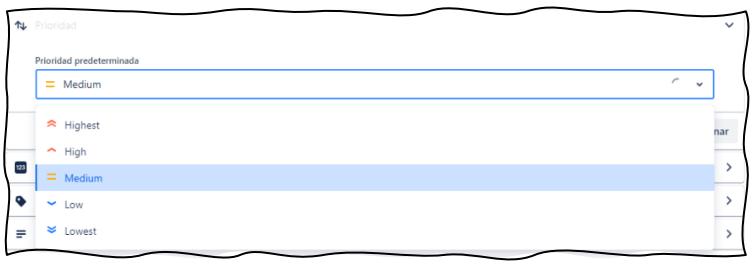

Figura 7 Prioridad Predeterminada en JIRA

Puntos de Historia: Cantidad de trabajo que requiere la historia de usuario.

Etiquetas:

- Categoría.
- Tipo de requisito (RequisitoFuncional, RequisitoNoFuncional, RequisitoProceso) donde:
  - ✓ Requisitos de Producto (necesidades o restricciones del producto):
    - Funcional. Describe cuál tiene que ser el comportamiento del sistema y los datos que debe tener en cuenta y/o almacenar.
    - No Funcional. Condiciones que debe cumplir el sistema, como aspecto de usabilidad y experiencia de usuario, escalabilidad, disponibilidad, fiabilidad, seguridad, rendimiento o mantenibilidad.
  - ✓ Requisito de Proceso (restricciones en el propio proceso de desarrollo de software como, por ejemplo, los costes del desarrollo).

Descripción: Detalla el comportamiento, restricciones, limitaciones y dependencias del requisito de manera que cumpla con la normativa. Inicialmente, este detalle será más genérico, pero cuando entre en un *Sprint*, se proporcionará una descripción más detallada.

Link: Asociación de documentación relevante para la historia de usuario, por ejemplo, documentos de diseño generados en *Confluence*.

Sprint: *Sprint* al que esta asignado el requisito.

Criterios de Aceptación: Dado que las pruebas se automatizarán, se definirá así: **Dado que** [precondición] **cuando** [acción] **entonces** [resultado esperado].

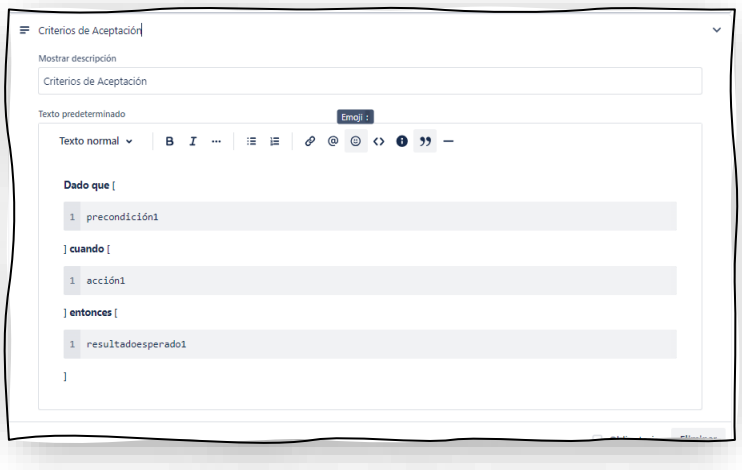

Figura 8 Configuración Criterios Aceptación en JIRA

Definition of Done (DoD): Criterios que definen cuando una historia ha sido finalizada.

- ☐ Está asociada a un EPIC
- ☐ Todas las tareas se han realizado
- ☐ Todos los criterios de aceptación cumplen el método SMART.
- ☐ Todos los criterios de aceptación se han automatizado.
- ☐ Se ha realizado la maqueta.
- ☐ Se ha realizado el diseño.
- ☐ Se ha mostrado en el *Sprint Review*.
- ☐ Cumple los criterios de Sonar.
- ☐ Se han aplicado correctamente las buenas prácticas.
- ☐ Ha superado con éxito el pipeline.
- ☐ El código ha sido revisado.

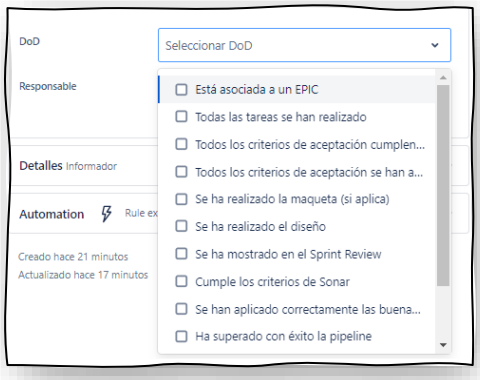

Figura 9 Lista del Definition of Done en JIRA

Desarrollo: Rama en la que se desarrollará la historia.

RESTRICCIÓN:

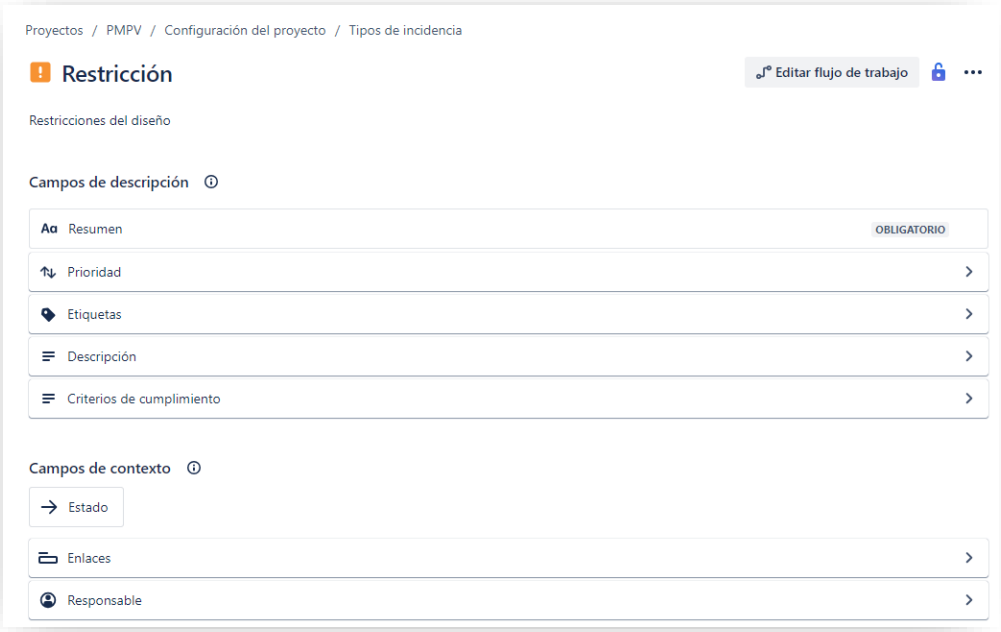

Figura 10 Plantilla Restricción en JIRA

Resumen: RS-Descripción breve de la restricción

Prioridad: Por defecto, alta.

Etiquetas:

- Tipo de requisito (RestricciónFuncional, RestricciónNoFuncional) donde:
  - ✓ Restricción Funcional. Describe limitaciones específicas en el comportamiento del sistema.

- ✓ Restricción No Funcional. Describe limitaciones relacionadas con aspectos de usabilidad, rendimiento, seguridad, etc.

Descripción: Detalla la descripción de la restricción e indica como afecta al sistema.

Criterios de Cumplimiento: Estándares que se deben cumplir para que la restricción sea satisfecha.

Link: Asociación de documentación relevante para la restricción.

Planificación del Sprint (Sprint Planning):

En la reunión de planificación del *Sprint*, el *Product Owner* presenta al equipo de *Scrum* las historias de usuario del *Product Backlog* que se abordarán. Después de aclarar las dudas que hayan surgido, el equipo de desarrollo utilizará la técnica del *Planning Poker* propuesta por *James Grenning* en 2002 para estimar de manera colaborativa los requisitos. Para ello, cada persona aportará su estimación y entre todos llegarán a un consenso.

Una vez definidos los puntos de historia de todas las *User Stories*, el *Product Owner,* considerando el ritmo de trabajo del equipo y los compromisos del equipo de desarrollo, incluye en el *Sprint Backlog* todas las *User Stories* que se llevarán a cabo durante el *Sprint.* En caso de conflicto en reunión de planificación, el *Scrum Master* actuará como mediador.

Ejecución del Sprint (Sprint Execution):

Durante la ejecución del *Sprint*, el equipo de desarrollo realiza las tareas de las *User Stories* incluidas en el *Sprint Backlog*. Además, se lleva a cabo una reunión de seguimiento diaria, conocida como *Daily Scrum*[6], donde cada miembro comparte lo que hizo el día anterior, lo que hará hoy y si tiene algún bloqueo. En caso de algún problema, el *Scrum Master* se encargará de resolverlo.

Revisión del Sprint (Sprint Review):

El equipo de desarrollo, el *Product Owner* y otros interesados se reúnen para verificar la correcta ejecución del trabajo realizado durante el *Sprint.* En la reunión, uno o varios miembros del equipo presentan las funcionalidades completadas y los interesados comparten sus observaciones y comentarios.

Retrospectiva del Sprint (Sprint Retrospective):

Este evento es clave para optimizar el proceso de desarrollo software e implica que el equipo *Scrum* reflexione sobre el *Sprint* que acaba de finalizar. Durante la reunión, cada miembro expone los aspectos positivos y negativos del desarrollo, se identifican posibles mejoras y se acuerdan acciones para el próximo *Sprint*.

[6] La reunión de seguimiento diaria será excluida de las actividades del Sprint por tratarse de un trabajo individual. Sin embargo, se realizará una valoración en el *Sprint Retrospective*.

2.6 Planificación y documentación de requisitos.

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 y administradores) 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.

Una vez establecido el marco teórico necesario y definido el sistema, se procede a la obtención inicial de requisitos. Para ello, se han recopilado los siguientes documentos: el plan de mantenimiento de un modelo de vehículo, proporcionado por un responsable de Toyota [Anexo 1], el manual de garantía de esta marca y el manual de propietario[7]. Además, se han tenido en cuenta los siguientes objetivos, pero el trabajo solo abordará los cinco primeros a lo largo de los cuatro *Sprints*.

- Análisis y diseño de la arquitectura.
- Gestión manual y automática de tareas de mantenimiento.
- Notificación a los ingenieros de modificaciones realizadas en tareas de mantenimiento.
- Gestión de usuarios.
- Gestión de roles y accesos.
- Gestión de planes de mantenimiento.
- Gestión de órdenes de mantenimiento.

[7] Se adjuntan solo los documentos públicos relativos al mantenimiento de Toyota, el resto, por derechos de autor, no han sido incluidos.

Historias de Usuario Priorizadas

En primer lugar, tras realizar un análisis inicial, se ha creado la lista de historias de usuario candidatas y se han priorizado. Estas listas se irán mostrando a lo largo de los diferentes *Sprints* y el resto se incluirán en el *backlog* del proyecto [Anexo 6].

Modelo de Dominio

El modelo de domino presenta una visión general del sistema a desarrollar, sin embargo, este trabajo solo abarcará la parte relativa a ingeniería[8] y usuarios. A continuación, se presenta el modelo de dominio de alto nivel de los requisitos candidatos.

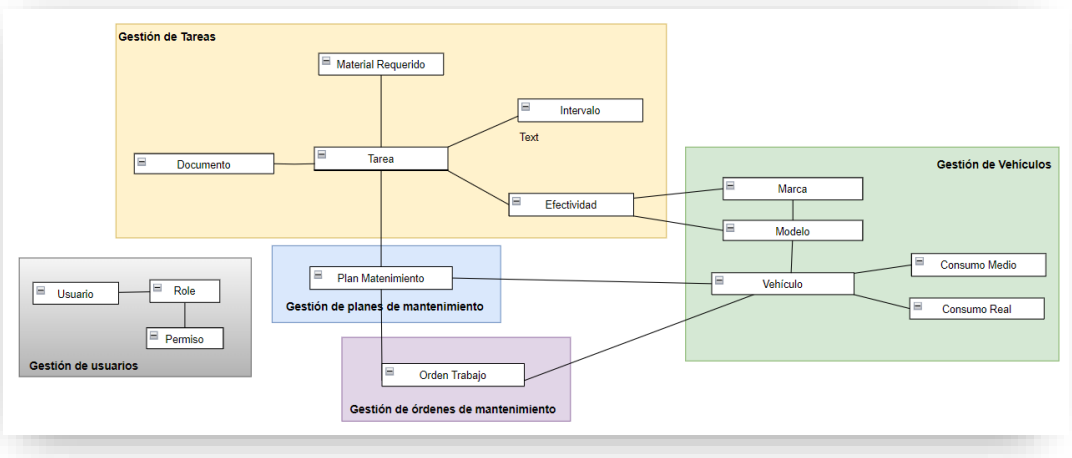

Figura 11 Modelo de Dominio de PMPV

Story Map

A continuación, se muestra el *User-Story Mapping* del primer *Sprint* para tener una visión más completa de este.

Figura 12 User Story Mapping - Sprint 0

[8] Gestión de tareas de mantenimiento.

Gestión de Riesgos

En cualquier proyecto pueden surgir problemas económicos, logísticos, de tiempo e incluso de coordinación, por lo tanto, es importante realizar una evaluación de riesgos y tener un plan de contingencia. A continuación, se muestra la evaluación de riesgos y el plan de contingencia de alguno de ellos.

Ref.	Nombre	Causa	Descripción	Consecuencia	Probabilidad	Impacto	Nivel
R01	Problemas en la entrega	Mayor complejidad de lo esperada	La estimación realizada no ha sido correcta.	Se produce un retraso en la entrega y aumentan los costos	Alta	Alto	Alto
R02	Problemas memoria	Cambios en la memoria	Se ha realizado un enfoque incorrecto.	Se rehace trabajo	Medio	Medio	Medio
R03	Problemas con las herramientas utilizadas	Herramientas utilizadas	Las herramientas utilizadas son complejas.	Se dedica más tiempo de lo esperado	Baja	Medio	Medio

Tabla 5 Evaluación de Riesgos

Código	Acción	Tipo	Riesgo Residual	Responsable	Fecha límite
A1R01	Reducir el número de historias que se realizarán debido a la complejidad	Corrector	Muy bajo	Product Owner	31/12/23
A1R02	Solicitar *feedback* al tutor más a menudo	Mitigadora	Medio	Responsable de la memoria	31/12/23
A1R03	Utilizar herramientas que se conozcan	Corrector	Bajo	Responsable de la documentación	31/12/23

Tabla 6 Medidas Correctoras

Presupuesto

Una vez se ha definido el alcance del proyecto y se han evaluado los posibles riesgos, se procede a la elaboración del presupuesto teniendo en cuenta las horas estimadas en el primer punto de la memoria. En este proceso, se establecerá la tarifa por hora considerando factores clave como: la experiencia del equipo, la industria y ubicación, la complejidad del proyecto, los recursos necesarios y una contingencia. Por lo tanto, teniendo en cuenta estos factores, se establece el costo de 20 euros por hora. Por lo tanto, el presupuesto inicial será el siguiente:

	Dedicación	Precio/h	Duración	Esfuerzo	Coste
PAC1					
Investigación	100%	20	10	30	600
Documentación	100%	20	4	12	240
PAC2					
Planificación Producto	100%	20	20	60	1200
Sprint 0	100%	20	22	66	1320
PAC3					
Sprint 1	100%	20	14	42	840
Sprint 2	100%	20	14	42	840
Sprint 3	100%	20	12	36	720
Memoria y Presentación					
Documentación	100%	20	17	51	1020
Presupuesto Base					6780
Contigencia (5%)					339
Total					**7119**

Figura 13 Presupuesto Inicial Proyecto

3. Sprint 0

3.1 Sprint Planning

En esta actividad, se estiman las *User Stories* que se llevarán a cabo durante el *Sprint*.

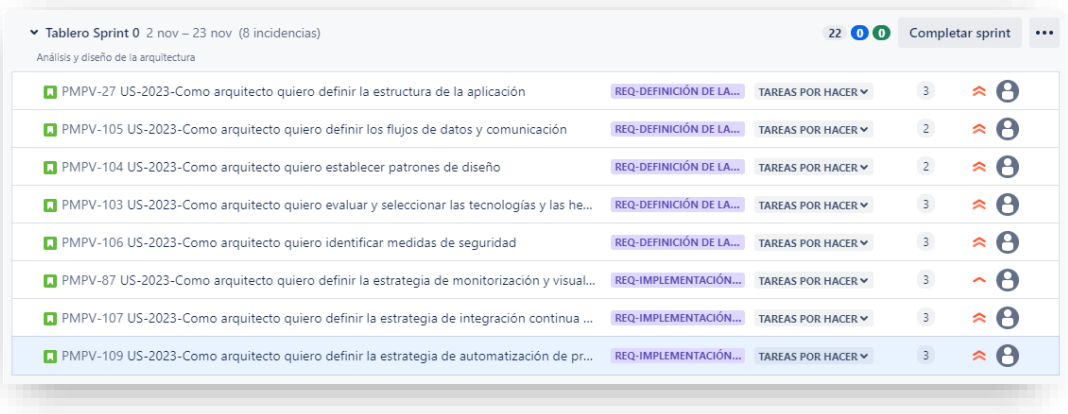

Figura 14 Planificación Sprint 0 en Jira

A continuación, se muestra la perspectiva de la *EPIC* relativa a la definición de la arquitectura donde están incluidas las restricciones.

Figura 15 EPIC Definición Arquitectura en Jira

3.2 Ejecución Sprint

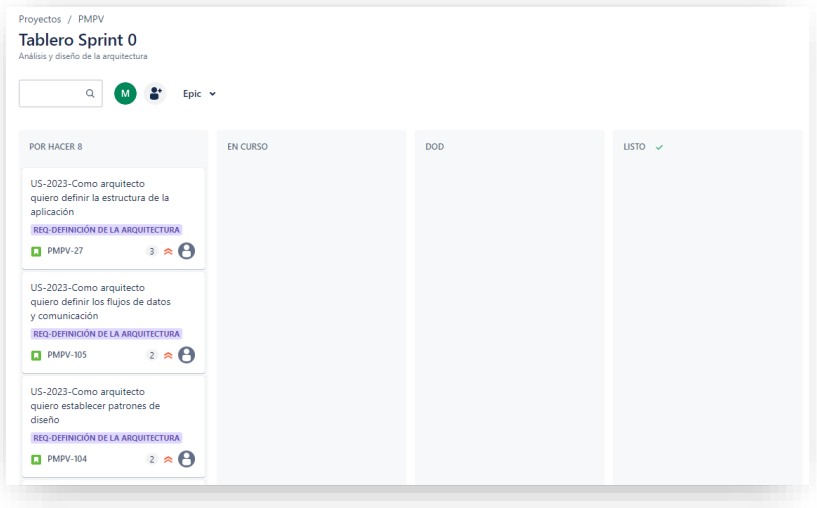

Figura 16 Tablero Sprint 0 en Jira

En este primer *Sprint*, se establecerá el diseño del *walking skeleton* [9] de la aplicación y para tener una visión global de la arquitectura del sistema, se han definido las cuatro dimensiones identificadas en el libro "*Fundamentos de la Arquitectura de Software" de Mark Richards y Neal Ford* [14]. Estas cuatro dimensiones, permiten reflejar las características de la arquitectura, las decisiones arquitectónicas, los principios de diseño y la estructura del sistema.

Características

En primer lugar, se extraen las características de la arquitectura a partir de los requisitos obtenidos en el capítulo anterior y que permiten cubrir los objetivos y necesidades de los *stakeholders*.

Características	**Detalle**
Seguridad	El sistema debe ser seguro y solo permitir acceso a usuarios autorizados.
Escalabilidad	El sistema debe ser escalable para solventar problemas de rendimiento en el futuro.
Tolerancia a fallos	El sistema debe ser tolerante a fallos para garantizar la disponibilidad y fiabilidad.
Separación por funcionalidades	El sistema debe separarse por funcionalidades para garantizar la escalabilidad, testeabilidad y mantenibilidad dada la complejidad de la aplicación.
Interoperabilidad	El sistema debe integrarse con otros sistemas externos.
	El sistema debe incluir la capacidad de realizar notificaciones.

Tabla 7 Características de la arquitectura

[9] Implementación mínima inicial de la arquitectura de la aplicación que incluye y conecta los componentes básicos del sistema.

Principios de diseño

A continuación, se establecerán los principios y pautas que posibilitarán la creación y organización de la arquitectura teniendo en cuenta las características que se han extraído de los requisitos.

Arquitectura

- Se debe mantener una documentación actualizada sobre la arquitectura.
- Es preferible realizar una división por funcionalidades.
- Es preferible la escalabilidad horizontal ya que facilita el crecimiento.
- Es preferible utilizar mensajes asíncronos entre los servicios ya que aumenta la tolerancia a fallos.

Interfaz de Usuario

- Es preferible que la interfaz de usuario sea intuitiva y reactiva.

Seguridad

- Es preferible que la comunicación entre los diversos servicios se realice a través de una *API Rest*, con interfaces y contratos claros y estandarizados.
- Es preferible incorporar medidas de seguridad que permitan gestionar los accesos a la aplicación.

Integración Continua y Despliegue Continuo (CI/CD)

- Es preferible que las diferentes funcionalidades de la aplicación implementen una funcionalidad específica, sean autónomas y tengan su propio proceso de integración continua (CI/CD).
- Es preferible que el proceso de integración continua (CI/CD) incluya las siguientes fases: compilación, pruebas, análisis de vulnerabilidades, calidad de código y despliegue.

Monitorización

- Es preferible monitorizar cada funcionalidad, centralizar la recopilación de datos y visualizarlos en tiempo real para su evaluación.

Automatización de Pruebas

- Es preferible realizar la automatización de las siguientes pruebas: unitarias, criterios de aceptación, regresión y reglas de gobernabilidad de la arquitectura[10].

Patrones de diseño [21]

- Arquitectura:
  - Patrón arquitectura de microservicios, permite diseñar una aplicación como una colección de servicios centrados en los dominios del producto proporcionando claridad y calidad en el código, facilita el

[10] Verificar que la implementación de código es consistente con los principios arquitectónicos definidos. Por ejemplo, si se han tenido en cuenta las buenas prácticas relatives a las dependencias entre los paquetes.

escalado, servicios independientes los unos de los otros y disponibles para ser comunicados en cualquier momento.

- Descomposición:
  - Patrón descomponer por subdominio, permite definir los servicios correspondientes a subdominios DDD. Este enfoque proporciona servicios cohesivos y débilmente acoplados, y equipos de desarrollo multifuncionales, autónomos y organizados en torno a la entrega del valor del producto.
  - Patrón servicio autónomo, permite, en colaboración con el patrón CQRS y Saga, responder a una solicitud sincrónica sin esperar la respuesta de ningún otro servicio. Este enfoque, aborda la disponibilidad de los servicios.

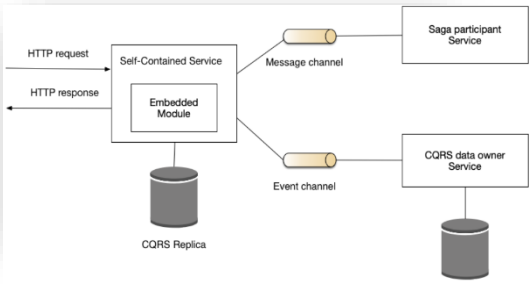

Figura 17 Solicitud síncrona sin espera [21]

- Gestión de datos:
  - Patrón base de datos por servicio. Cada servicio tiene su propia base de datos privada, lo que permite que estén débilmente acoplados y utilicen la BBDD que mejor se adapte a sus necesidades. En este enfoque, será necesario aplicar los patrones CQRS [11] y Saga para implementar transacciones y consultas que abarquen servicios.
  - Patrón CQRS, separa la parte de comandos de la de consultas. Este enfoque simplifica los modelos de consulta y comando.
  - Patrón Saga, permite mantener la consistencia de datos a lo largo de los servicios.
- Pruebas:
  - Patrón prueba de contrato impulsada por el consumidor, permite verificar a un servicio que las llamadas a otro servicio cumplen con sus expectativas.
  - Patrón prueba de componente de servicio, permite verificar un servicio de forma aislada.
- Despliegue:
  - Patrón instancia de servicio por contenedor, empaqueta y ejecuta el servicio usando contenedores *Docker* independientes.
- API externo:
  - Patrón *API Gateway*, permite ofrecer una pasarela que sirve de punto único de acceso a la aplicación encapsulando toda la arquitectura interna, de modo que no se expongan los datos y operaciones.

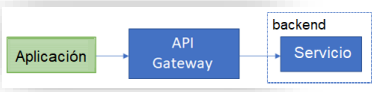

Figura 18 API Gateway

[11] Segregación de responsabilidades de consultas.

- Seguridad:
  - Patrón token de acceso, permite comunicar la identidad del solicitante a los servicios que gestionan la solicitud. Este enfoque garantiza la seguridad y autenticación de las solicitudes, es decir, *API Gateway* autentica la solicitud y pasa un *token* de accesos que identifica de forma segura.

- Comunicación:
  - Patrón mensajería, permitirá enviar mensajes a otros servicios de forma asíncrona y para garantizar la consistencia se utilizará con el patrón bandeja de salida transaccional. Este enfoque permite la colaboración entre los servicios de forma asíncrona. Por ejemplo, *Kafka*.
  - Patrón bandeja de salida transaccional, permite garantizar la consistencia de datos ya que el mensaje solo se envía si la transacción se ha completado correctamente.

- Observabilidad[12]:
  - Patrón agregación de registros, permite centralizar la información registrada en un servicio.
  - Patrón API de comprobación de estado, permite detectar el mal funcionamiento de una solicitud.
  - Patrón métricas de aplicación, permite recopilar métricas específicas de una aplicación.

Decisiones

En esta dimensión, se expondrán las decisiones que se han tomado para implementar la arquitectura especificando tecnologías, plataformas y protocolos de comunicación.

Documentación

- Realizar la documentación en *Confluence* para mantener la trazabilidad con *Jira* y que todo esté centralizado en un repositorio.
- Los *API Rest* se documentarán en *Swagger*.

Estrategia de versionado

Adoptar el Versionado Semántico [19] para gestionar las versiones de la aplicación con la convención de número de versión MAYOR.MENOR.PARCHE que se incrementa:

- La versión MAYOR cuando realizas un cambio incompatible en el API.
- La versión MENOR cuando añades funcionalidad compatible con versiones anteriores.
- La versión PARCHE cuando reparas errores compatibles con versiones anteriores.

Esta estrategia, mejora la claridad de los cambios en las diferentes versiones.

[12] Comprender y monitorizar el comportamiento interno de un sistema en tiempo real a través de la recopilación de datos.

Estrategia de ramificación (*branching*)

Aplicar el método *Git Flow* planteado por *Vincent Driessen en 2010* [20] para proporcionar una gestión de ramas clara y organizada que facilite el trabajo del equipo de desarrollo.

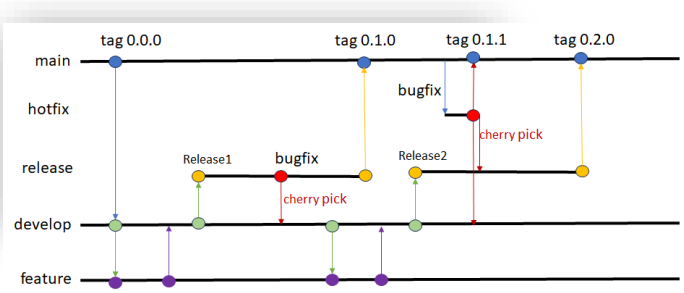

Figura 19 Git Flow & Versionado

Tecnologías backend

- *Spring Framework* y *Spring Boot*: Marcos de trabajo que facilitan el desarrollo con inyección de dependencias y aseguran el correcto funcionamiento de las solicitudes.
- *Postgress*: Base de datos que asegura la integridad de datos ACID[13] y admite consultas relacionales y no relacionales.
- *Apache Kafka*: Para mensajería asíncrona y procesamiento de eventos con tolerancia a fallos. Además, es perfecto para sistemas distribuidos.
- *Zookeeper*: Servido de código abierto que permite coordinar de manera fiable y consistente procesos distribuidos.

Tecnologías frontend

- *React*: Marco de trabajo que permite crear interfaces de usuario intuitivas y reactivas, fácil de aprender con conocimientos de *JavaScript*.
- *Bootstrap:* Librería de estilos basado en HTML y CSS que facilita el diseño *responsive*.

Infraestructura de despliegue

- *Jenkins*: Servidor enfocado en la automatización de procesos de integración continua, es de código abierto, permite utilizarlo en conjunto con *Docker* y se configura fácilmente.
- *SonarQube*: Herramienta de código abierto que garantiza la calidad del código analizando errores, vulnerabilidades y malas prácticas de programación.
- *Azure*: Proveedor en la nube en el que es fácil cargar imágenes *Docker*, automatizar el proceso con *Jenkins* y levantar sus instancias.
- *Docker*: Permite la contenerización de los servicios proporcionando servicios aislados y portátiles.
- *Kubernetes*: Permite orquestar y gestionar eficientemente las instancias de los contenedores y se integra fácilmente con *Azure*.

[13] ACID: Atomicidad, consistencia, aislamiento y durabilidad.

Monitorización
- *Micromiter*: Recopila métricas del servicio para evaluarlo.
- *Prometeus*: Herramienta de código abierto que almacena de forma centralizada las métricas.
- *Grafana*: Herramienta web de código abierto que permite configurar y visualizar las métricas.

Pruebas automáticas
- *Selenium:* Permite escribir y gestionar pruebas de aceptación automatizadas en proyectos basados en Java.
- *Concordion*: Herramienta de código abierto que se utiliza para la especificación y la ejecución de pruebas de aceptación automatizadas.
- *Junit5*: Marco de trabajo de pruebas unitarias para Java que permite verificar de forma automática la aplicación.
- *ArchUnit*: Librería que permite verificar la estructura del código.

Protocolos de comunicación
- *HTTP y API Rest*: para la comunicación entre los servicios.
- *API Gateway*: para ofrecer un único punto de acceso a la aplicación.

Seguridad:
- *Spring Security* como marco de seguridad para la autorización y autenticación de la aplicación por su robustez, sencillez y flexibilidad en la protección de aplicaciones web y de servicios en *Spring*.

Por último, hay que indicar que, utilizando *Kafka*, *Docker* y *Kubernetes* se garantiza la escalabilidad. Además, *Kafka* y *Zookeeper* proporciona tolerancia a fallos.

Estructura

Esta última dimensión permitirá obtener una visión de la organización y disposición de los componentes en el sistema, incluyendo el estilo arquitectónico, la organización de los módulos y las conexiones entre ellos.

En primer lugar, el sistema seguirá un estilo arquitectónico basado en microservicios y cada servicio adoptará un estilo Hexagonal[14]. Esto permitirá una clara separación de responsabilidades. Además, se utilizará *Docker* para contenerizar cada servicio, de modo que su mantenimiento y despliegue sean más sencillos.

Cada servicio representará un subdominio de la aplicación y tendrá su propia base de datos, sistema de monitorización y una integración continua con pruebas propias. Además, proporcionará acceso a sus funcionalidades a través de *APIs*, y mediante *Docker Compose*, facilitará la comunicación entre los servicios a través de una red compartida entre ellos.

[14] La arquitectura hexagonal organiza la lógica en el centro, rodeada de puertos y adaptadores, permitiendo una clara separación entre la lógica de negocio y los componentes externos.

Para gestionar el acceso a las funcionalidades de los servicios, se implementará una fachada que los expondrá a través de una *API Gateway* con autenticación. Esto proporcionará un punto de entrada único y asegurará que los servicios estén aislados del exterior. A continuación, se muestra la estructura completa mediante este diagrama.

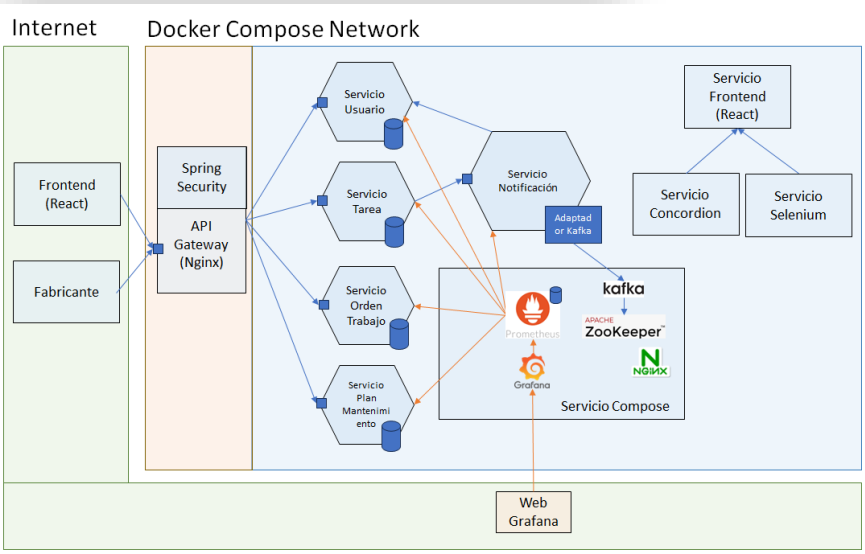

Figura 20 Diagrama de componentes de la aplicación

Finalmente, cabe destacar que la infraestructura de producción estará alojada en Azure y se habilitará un acceso SSH para conexiones externas. Además, el despliegue se realizará a través de una integración continua con Jenkins que proporcionará una imagen de cada uno de los servicios.

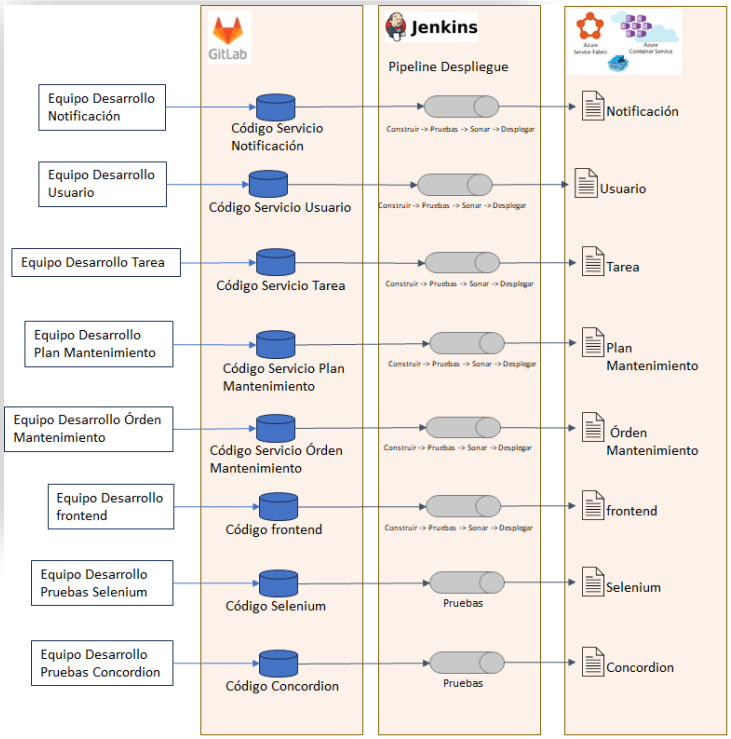

Figura 21 Integración continua de los servicios

3.3 Sprint Review

Esta actividad, se centra en demostrar cómo se han seguido las buenas prácticas en la documentación del análisis y diseño de la arquitectura en *Jira* y *Confluence* a lo largo del *Sprint*.

En primer lugar, para satisfacer las buenas prácticas de CMMI, se han reutilizado plantillas existentes en *Confluence* y, también, se han creado nuevas. Por ejemplo, se ha visto la necesidad de generar una para la retrospectiva y otra para la *Review*.

Figura 22 Plantillas para Review y Retrospective

Posteriormente, para cumplir con la norma ISO/IEC/IEEE 12207:2017, el documento de análisis y diseño de la arquitectura

[Anexo 2] ha integrado las recomendaciones indicadas en la *Tabla 4 Cumplimiento de resultados ISO/IEC/IEEE 12207:2017 con SCRUM*. Esto abarca aspectos como los interesados, trazabilidad con los requisitos y diagramas necesarios.

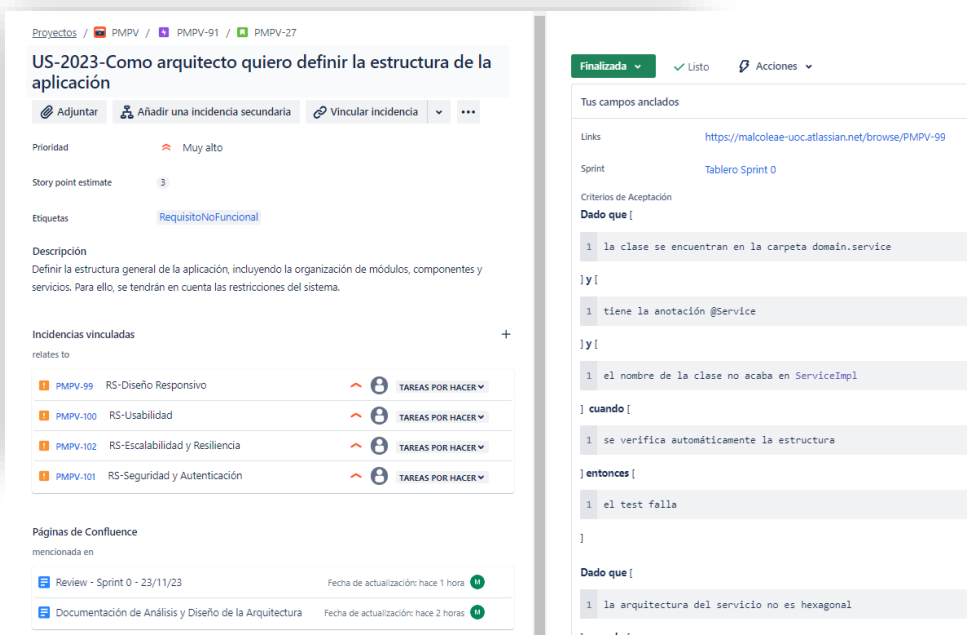

Figura 23 Review - Sprint 0 – User Story Finalizada

Finalmente, *Confluence* ha facilitado enormemente la trazabilidad con los requisitos y ha permitido añadir nueva documentación a través de importaciones de una manera fácil. Del mismo modo, es posible exportarla.

3.4 Sprint Retrospective

La retrospectiva se ha realizado en Miro y documentado en Confluence:

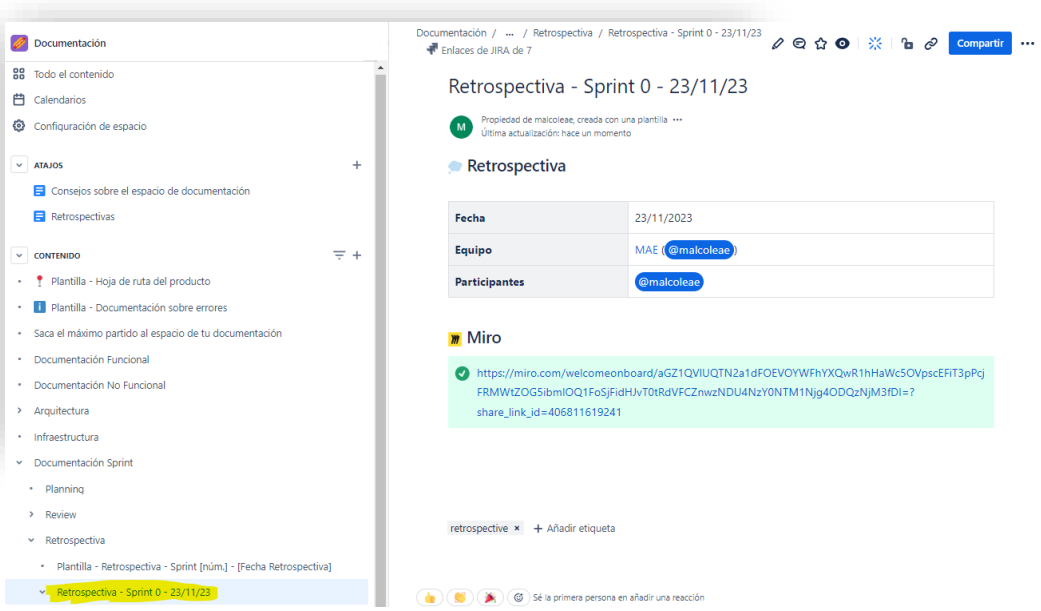

Figura 24 Retrospectiva - Sprint 0 – Documentación Confluence

En primer lugar, cada miembro del equipo pone un *post-it* en la imagen que mejor representa su estado de ánimo durante el *Sprint* e indica el motivo por el cual se ha sentido así. Esto proporciona una visión del estado emocional del equipo.

Figura 25 Retrospectiva - Sprint 0 – Miro – Emociones

En esta retrospectiva, se han seleccionado los rayos y truenos porque se ha realizado un esfuerzo adicional en la entrega de la PEC 2. Se ha proporcionado más contenido del previsto en planificación para poder proporcionar una visión más clara del análisis y diseño de la arquitectura, evitando que quedara incompleto. Este hecho, ha tenido un ligero impacto en el presupuesto inicial del proyecto debido a la realización de 9 horas más de las estimadas, lo que equivale un incremento en el presupuesto inicial de 180 euros.

Posteriormente, cada persona expone lo mejor del Sprint, lo que ha facilitado el trabajo, los obstáculos encontrados y los posibles riesgos. Esto proporciona una visión clara de lo que está funcionando y de las cosas que hay que mejorar.

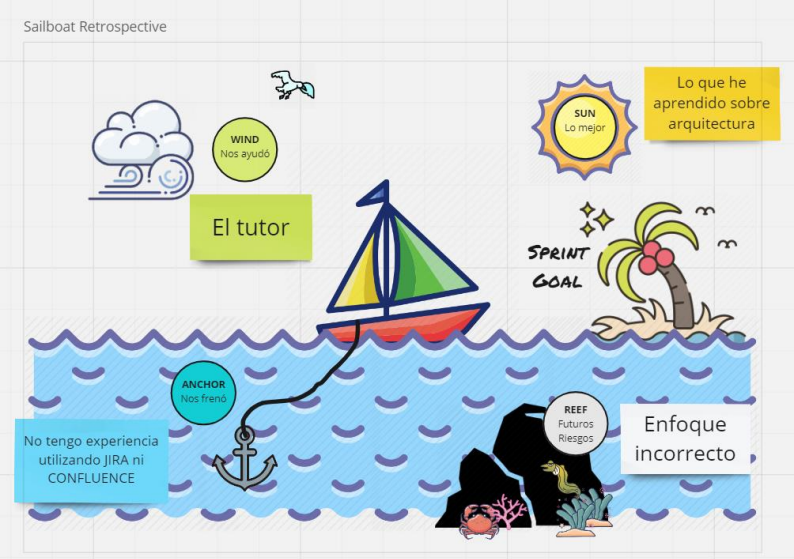

Figura 26 Retrospectiva - Sprint 0 – Miro – Comunicación

En este caso, lo mejor del *Sprint* ha sido el conocimiento que se ha adquirido en el análisis y diseño de la arquitectura gracias a las cuatro dimensiones establecidas por Mark Richards y Neal Ford. Además, de la ayuda proporcionada por el tutor.

Sin embargo, se ha tardado algo más de tiempo en llevar a cabo la documentación en *Jira* y *Confluence* debido a la falta de experiencia en estas dos herramientas. Además, se ha visto un posible riesgo en cómo se han enfocado las cuatro dimensiones por la complejidad de definir los límites entre ellas.

Finalmente, los miembros del equipo votan por uno de los problemas expuestos en la retrospectiva y se comprometen a tomar acciones durante el siguiente *Sprint*, contribuyendo así a la mejora continua del proceso.

En este *Sprint*, se realiza el compromiso de ponerse en contacto con el tutor más a menudo para recibir *feedback* y minimizar el riesgo de un enfoque incorrecto en la documentación.

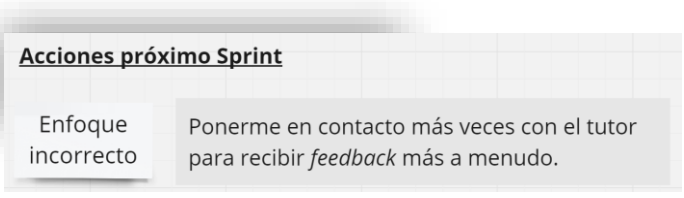

Figura 27 Retrospectiva - Sprint 0 – Miro – Acciones

4. Sprint 1

4.1 Sprint Planning

El objetivo de este *Sprint* es realizar el análisis y el diseño relativo a la gestión de usuarios, la gestión de roles, asignar y quitar permisos a un rol y, finalmente, la autenticación en el sistema.

Para ello, las *User Stories* proporcionadas por el *Product Owner* se han estimado y, posteriormente, aquellas a las que el equipo de desarrollo se ha comprometido, se han incluido en el *backlog* del *Sprint*.

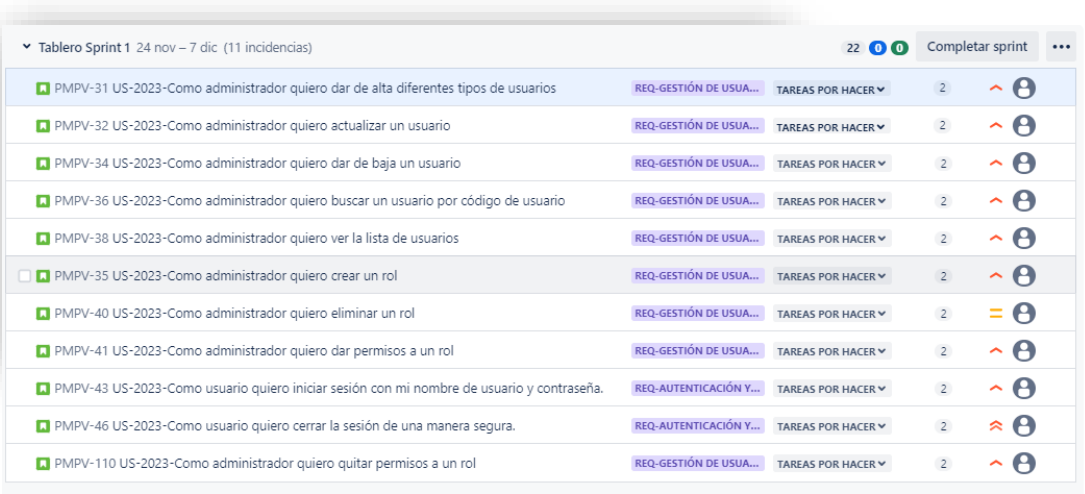

Figura 28 Planificación Sprint 1 en Jira

A continuación, se muestra el detalle de la *User Story* relativa a la creación de un usuario en el sistema, por parte del administrador, teniendo en cuenta los estándares.

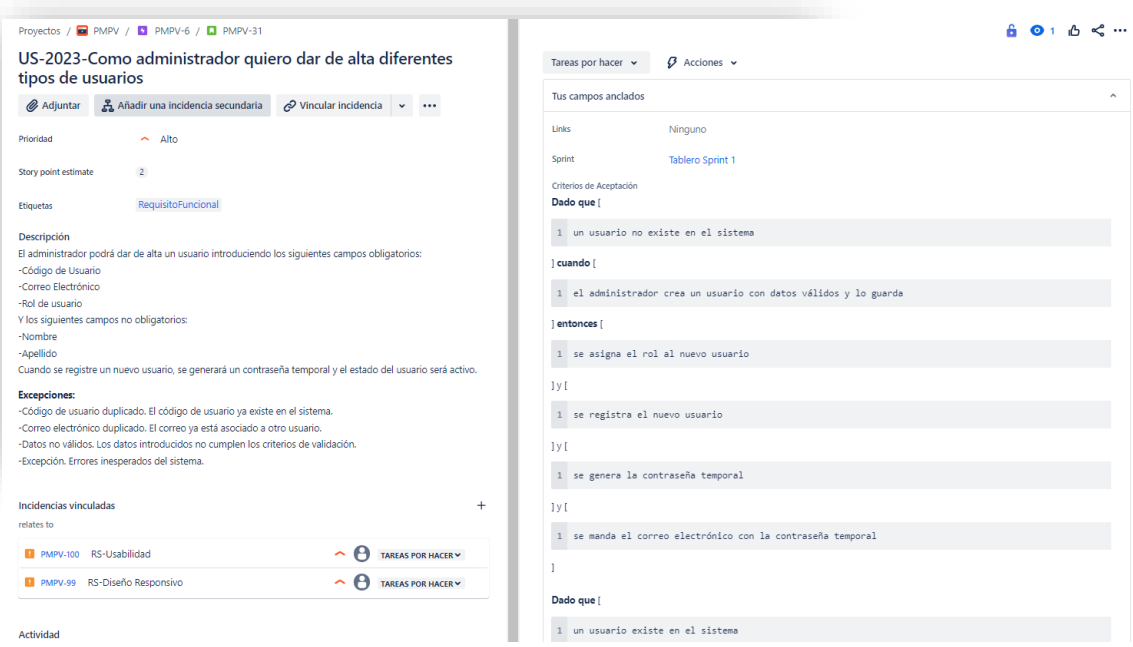

Figura 29 User Story Creación Usuario en Jira

Esta *User Story* aborda los siguientes **criterios de aceptación** para garantizar el cumplimiento del requisito funcional.

1 **Dado que** un usuario no existe en el sistema **cuando** el administrador crea un usuario con datos válidos y lo guarda **entonces** se asigna el rol al nuevo usuario **y** se registra el nuevo usuario **y** se genera la contraseña temporal.

2 **Dado que** un usuario existe en el sistema **cuando** el administrador intenta crear el mismo usuario **entonces** se muestra un mensaje de error indicando “El usuario ya se ha dado de alta en el sistema” **y** el usuario no se registra en el sistema.

3 **Dado que** un usuario existe en el sistema **cuando** el administrador intenta crear el mismo usuario **entonces** se muestra un mensaje de error indicando “El usuario ya se ha dado de alta en el sistema” **y** no se registra el usuario.

4 **Dado que** el campo de correo no se ha introducido **cuando** el administrador intenta crear el usuario **entonces** se muestra un mensaje de error indicando "El correo electrónico es obligatorio" **y** el usuario no se registra en el sistema.

5 **Dado que** el campo del código de usuario no se ha introducido **cuando** el administrador intenta crear el usuario **entonces** se muestra un mensaje de error indicando "El código de usuario es obligatorio" **y** el usuario no se registra en el sistema.

6 **Dado que** el campo del rol no se ha introducido **cuando** el administrador intenta crear el usuario **entonces** se muestra un mensaje de error indicando "El rol del usuario es obligatorio" **y** el usuario no se registra en el sistema.

7 **Dado que** el campo de correo electrónico no tiene la @ **cuando** el administrador intenta crear el usuario **entonces** se muestra un mensaje de error indicando "El correo electrónico es incorrecto" **y** el usuario no se registra en el sistema.

4.2 Ejecución Sprint

Permisos y Seguridad

En primer lugar, se definen los niveles de permisos para los distintos roles y se establecen las medidas de seguridad en el sistema. Esto garantiza la seguridad en la autenticación y autorización de la aplicación, de modo que un usuario autenticado solo acceda a aquellas áreas permitidas. A continuación, se detallan los diferentes niveles de acceso que habrá por defecto, ya que el administrador podrá configurarlo.

Módulo	Admin	Operador	Ingeniero	Mecánico
Gestión de usuarios	☑	☐	☐	☐
Gestión de roles	☑	☐	☐	☐
Gestión de permisos	☑	☐	☐	☐
Gestión de tareas	☐	☐	☑	☐
Gestión de plan de mantenimiento	☐	☐	☑	☐
Gestión de vehículos	☐	☑	☐	☐
Gestión de órdenes de mantenimiento	☐	☐	☐	☑

Tabla 8 Niveles de permisos

Después de exponer los niveles de acceso, es esencial comprender cómo se diseña la seguridad. Para facilitar este proceso, se empleará la seguridad de *Spring Boot* y *JSON Web Tokens* (JWT). Inicialmente, un JWT actúa como un

token de autenticación y no de autorización, verificando la identidad del usuario, pero sin proporcionar acceso directo a los recursos solicitados. Sin embargo, es importante resaltar que cuando un JWT contiene claims[15], proporciona información de autorización además de autenticación.

Con el objetivo de facilitar la transferencia segura y eficiente de la información de autenticación y autorización entre diferentes sistemas a través de la web, se utiliza JWS[16]. Un token en formato JSON generado por JWS contiene la información estructurada de la siguiente manera:

- Encabezado: información del tipo de token y del algoritmo de firma.
- Cuerpo: contiene los datos de la sesión, usuario, roles y permisos.
- Firma: para verificar la integridad del token.

Por lo tanto, utilizando el patrón API Gateway y el patrón Token de Acceso proporcionado por *Spring Security*, las personas autorizadas pueden acceder de manera segura a las funcionalidades correspondientes. Esto se logra gracias a la firma digital, así como a la inclusión de los roles y permisos del usuario en el token. A continuación, se ilustra su arquitectura:

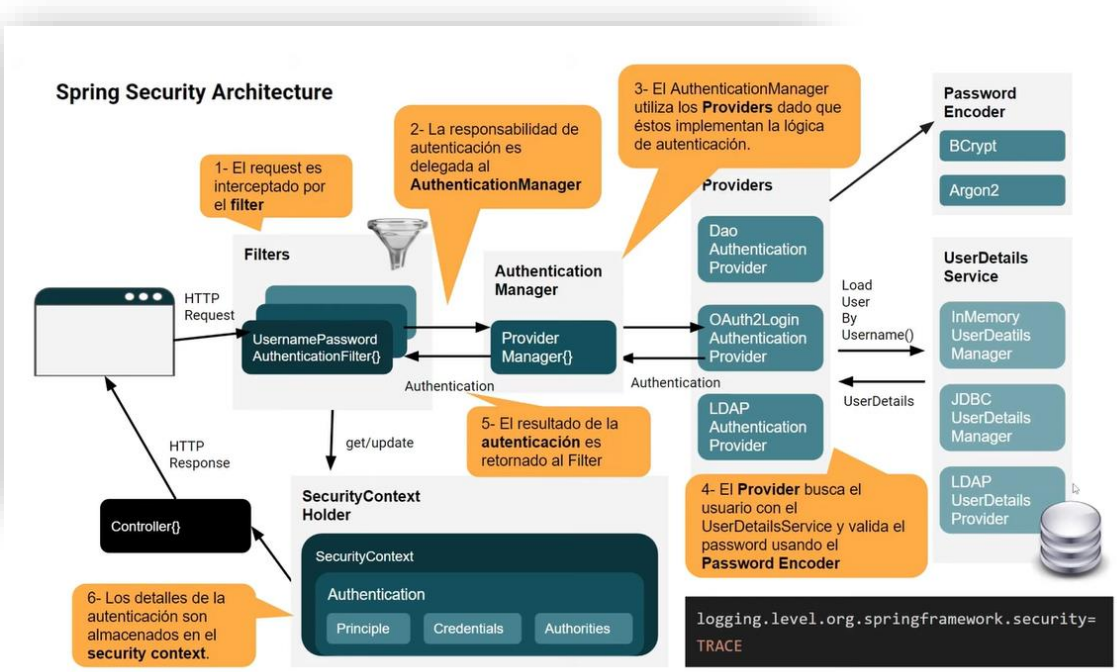

Figura 30 Arquitectura de Spring Security [33]

Tal y como se puede observar, antes de llamar al controlador, el filtro de autenticación verifica las credenciales de la sesión a través del proveedor OAuth2Login y los datos obtenidos del usuario. Una vez autenticado el usuario, se almacena en el contexto de seguridad de *Spring Boot* y se envía al cliente

[15] Un claim proporciona información relativa a un usuario, permisos, roles y otros datos relevantes que se agrupan y certifican mediante un token de seguridad.
[16] JSON Web Signature (JWS) garantiza la integridad del contenido mediante firmas digitales o códigos de autenticación de mensajes en formato JSON.

para que este pueda utilizarlo en aquellas solicitudes que estén protegidas y requieran autenticación.

A continuación, se muestra el proceso de autenticación utilizando *Spring Security* y *JSON Web Tokens* (JWT), donde el usuario se loga en el sistema y accede al área del sistema en el que tiene acceso.

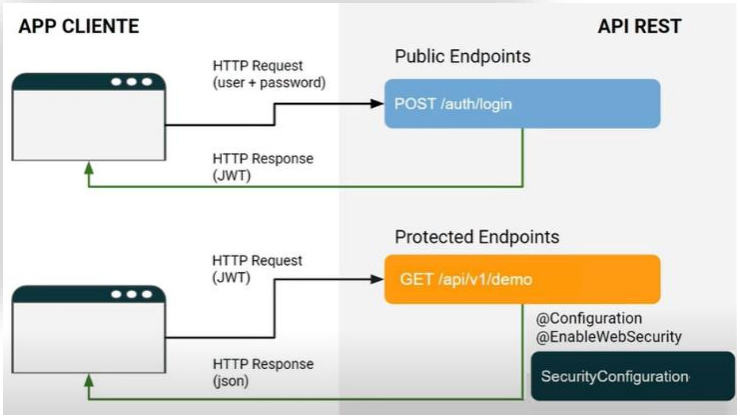

Figura 31 Autenticación Spring Security + JWT [32]

Un punto a destacar es la importancia de configurar adecuadamente el intercambio de recursos de origen cruzado (CORS) para permitir solicitudes desde diferentes dominios, ya que es esencial para prevenir posibles problemas de acceso y garantizar un intercambio seguro y eficiente de la información entre sistemas de dominios diferentes.

Modelo de Dominio

Una vez se han especificado los diferentes permisos, se establece el modelo de dominio del subdominio relativo a la gestión de usuarios que se corresponderá con el servicio Usuario. Para ello, se tendrán en cuenta principalmente los nombres de las historias de usuario. Esto permitirá definir las operaciones del sistema.

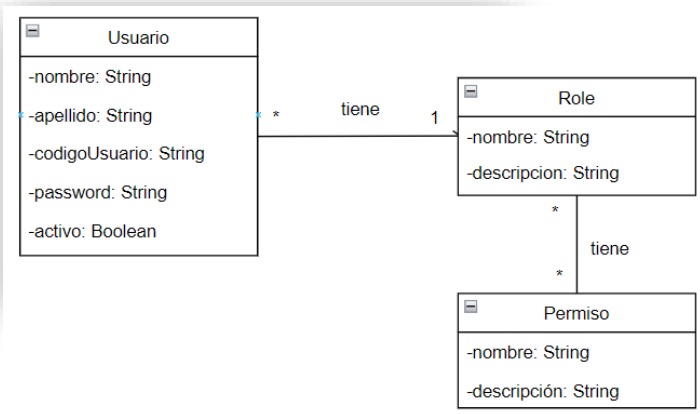

Figura 32 Modelo de dominio del servicio de usuario

Operaciones del Sistema

Una vez definido el modelo de dominio, se identifican las solicitudes que podrá realizar un usuario, a estas solicitudes se les llamará operaciones y, en este caso, se tendrán en cuenta los verbos. Dado que se aplica el patrón CQRS, para simplificar el modelo, se tendrán dos tipos de operaciones: comandos y consultas. Además, estas operaciones se corresponderán con los endpoints de la aplicación.
Comandos:

Actor	Caso de uso	Nombre comando	Descripción
Administrador	Crear usuario	*crearUsuario()*	Crear un usuario activo que acceda a la aplicación.
Administrador	Modificar usuario	*modificarUsuario()*	Modificar los datos del usuario.
Administrador	Dar Baja Usuario	*bajaUsuario()*	Cambiar el estado de activo a inactivo.
Sistema	Generar Contraseña Temporal	*generarPasswordTemporal()*	Generar una contraseña.
Administrador	Crear Rol	*crearRol()*	Crear un rol.
Administrador	Eliminar Rol	*eliminarRol()*	Eliminar un rol.
Administrador	Asignar Permiso Rol	*asignarPermisoRol()*	Asignar un permiso a un rol.
Administrador	Quitar Permiso Rol	*quitarPermisoRol()*	Quitar un permiso a un rol.

Tabla 9 Comandos Gestión de Usuarios

Consultas:

Actor	Caso de uso	Nombre comando	Descripción
Administrador	Ver Lista Usuarios	*consultarUsuarios()*	Recupera la información de los usuarios.
Administrador	Buscar Usuario por código	*BuscarUsuarioPorCodigo()*	Recupera la información de un usuario por el código de usuario.
Administrador	Ver Lista Roles	*consultarRoles()*	Recuperar los roles.

Tabla 10 Consultas Gestión de Usuarios

A continuación, se muestra el diagrama de casos de uso que permite visualizar más claramente la iteración entre el administrador y el sistema.

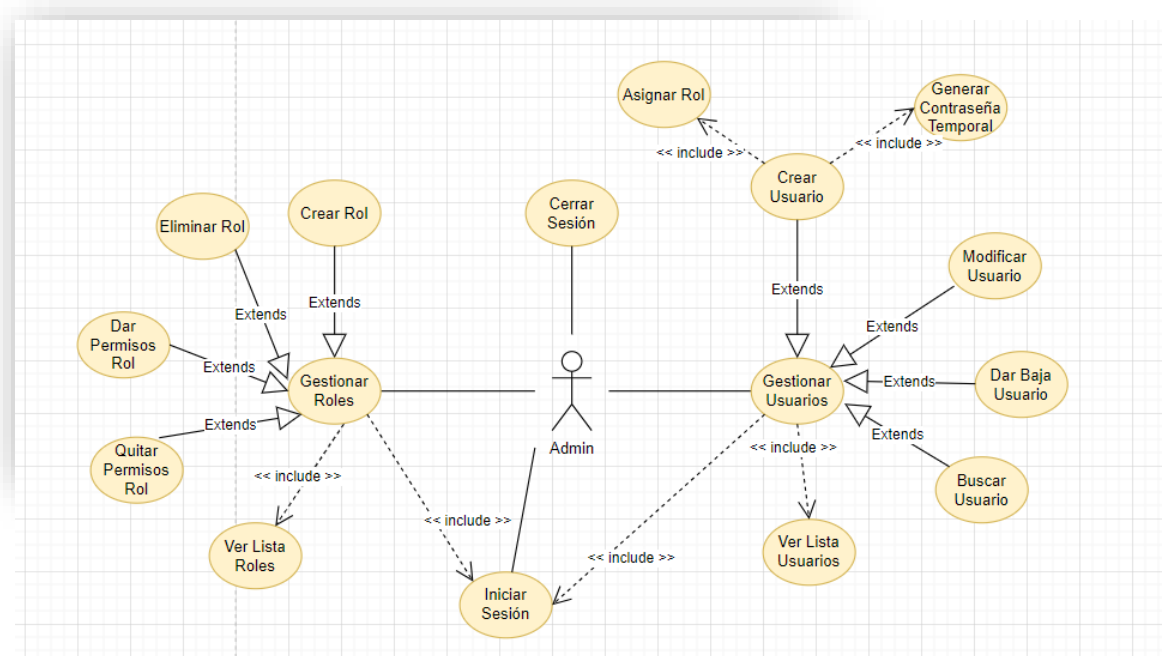

Figura 33 Diagrama de casos de uso de la gestión de usuarios y roles

Especificación de los comandos

En la especificación se definen los parámetros del comando, los valores que retornará y el comportamiento que tendrá, es decir, las precondiciones y postcondiciones de la clase.

Operación	***crearUsuario***(nombre, apellido, codigoUsuario, *password*, email, rolId)
Retorno	usuarioId
Precondiciones	▪ El usuario no existe en el sistema. ▪ El rol existe en el sistema.
Post-condiciones	▪ El usuario se crea con estado ACTIVO. ▪ La contraseña se almacena encriptada.

Operación	***bajaUsuario***(usuarioId)
Retorno	True/False
Precondiciones	▪ El usuario existe en el sistema. ▪ El estado del usuario es ACTIVO.
Post-condiciones	▪ El estado del usuario se cambia a INACTIVO.

Operación	***generarPasswordTemporal***()
Retorno	passwordTemporal
Precondiciones	-
Post-condiciones	▪ El sistema generara una *password* temporal.

Operación	***crearRol***(nombre, descripción)
Retorno	rolId
Precondiciones	▪ El rol no existe en el sistema
Post-condiciones	▪ El rol se crea en el sistema.

Operación	***eliminarRol***(rolId)
Retorno	True/False
Precondiciones	▪ El rol existe en el sistema. ▪ El rol no está asignado a ningún usuario.
Post-condiciones	▪ Los permisos del rol se quitan de la lista del rol. ▪ El rol se elimina del sistema.

Operación	***asignarPermisoRol***(rolId, permisoId)
Retorno	True/False
Precondiciones	▪ El rol existe en el sistema. ▪ El permiso existe en el sistema.
Post-condiciones	▪ Se añade el permiso a la lista de permisos del rol.

Operación	***quitarPermisoRol***(rolId, permisoId)

Retorno	True/False
Precondiciones	▪ El rol existe en el sistema. ▪ El permiso existe en el sistema.
Post-condiciones	▪ Se elimina el permiso de la lista de permisos del rol.

Tabla 11 Especificación de comandos de gestión de usuarios

Base de Datos

Un aspecto clave del diseño es la relación entre los diferentes elementos que conforman la gestión de usuarios. Por lo tanto, se proporciona el diagrama relacional que muestra las relaciones entre los diferentes elementos permitiendo la gestión de usuarios roles y permisos de manera efectiva.

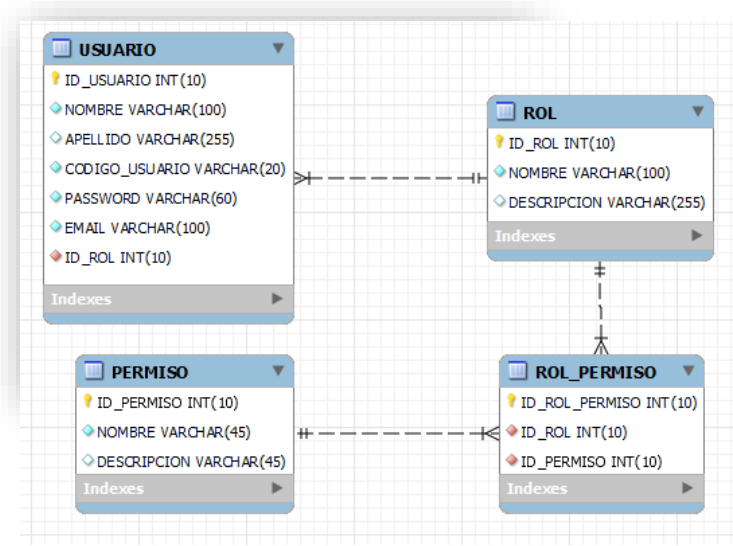

Figura 34 Diagrama relacional del modelo de datos de la gestión usuarios

Por defecto, el sistema, teniendo solo en cuenta la gestión de usuarios, tendrá los siguientes permisos:

Gestión de usuarios	CREAR_USUARIO	Permite crear nuevos usuarios.
	MODIFICAR_USUARIO	Permite modificar información de usuarios.
	VER_ USUARIO	Permite ver información de usuarios.
	ELIMINAR_USUARIO	Permite dar de baja usuarios.
Gestión de roles	CREAR_ROL	Permite crear nuevos usuarios.
	MODIFICAR_ ROL	Permite modificar información de usuarios.
	VER_ ROL	Permite ver información de usuarios.
	ELIMINAR_ ROL	Permite dar de baja usuarios.
Gestión de permisos	ASIGNAR_PERMISO	Permite asignar permisos a un rol.
	QUITAR_PERMISO	Permite quitar permisos a un rol.

Tabla 12 Permisos de la Gestión Usuarios

Autenticación y Autorización

Del mismo modo, es importante comprender la iteración entre los componentes que formarán parte del proceso de autenticación y autorización, anteriormente

explicado en detalle. De modo que, para facilitar su comprensión, se proporciona el siguiente diagrama de secuencia.

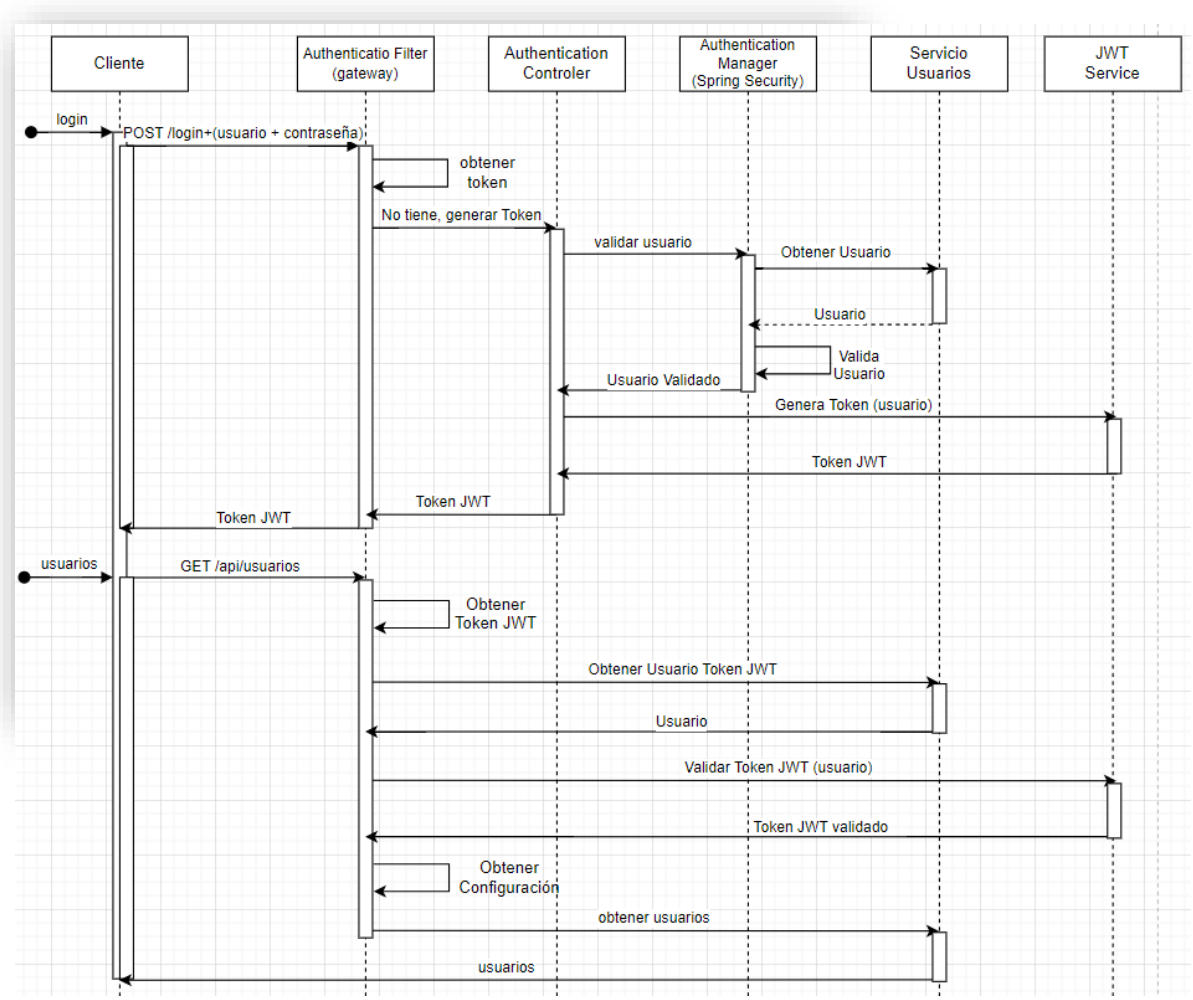

Figura 35 Diagrama de secuencia de autenticación y autorización

Interfaz de usuario

Finalmente, una vez clara la funcionalidad del sistema, se diseña la interfaz de usuario. Para ello, se consideran las restricciones impuestas por el *Product Owner*: la usabilidad del sistema debe ser intuitiva, fácil de usar y *responsive.* Estas directrices garantizarán una gran experiencia de usuario.

Con el objetivo de lograrlo, se ha realizado un prototipo de baja fidelidad en *Figma*[17] en el que se cumplen los siete principios del diseño universal[18].

[17] *Figma* es una herramienta que genera prototipos.
[18] Ronald L. Mace acuñó el término diseño universal para referirse al tipo de diseño que cualquier persona puede entender y usar. Este diseño define siete principios que se tienen que cumplir: igualdad de uso, flexibilidad, simple e intuitivo, información fácil de percibir, tolerante a errores, escaso esfuerzo físico y dimensiones apropiadas.

La elección de que sea de baja fidelidad es porque es fácil de crearlo y permite a los interesados evaluar la interfaz antes de realizar el desarrollo. A continuación, se muestran las pantallas del listado de usuarios y del inicio de sesión.

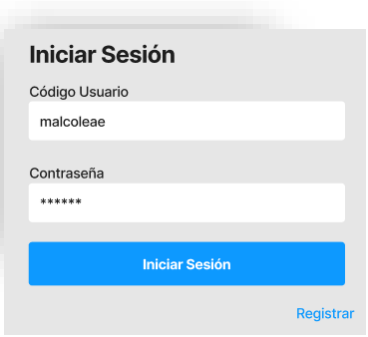

Figura 36 Pantalla del inicio de sesión

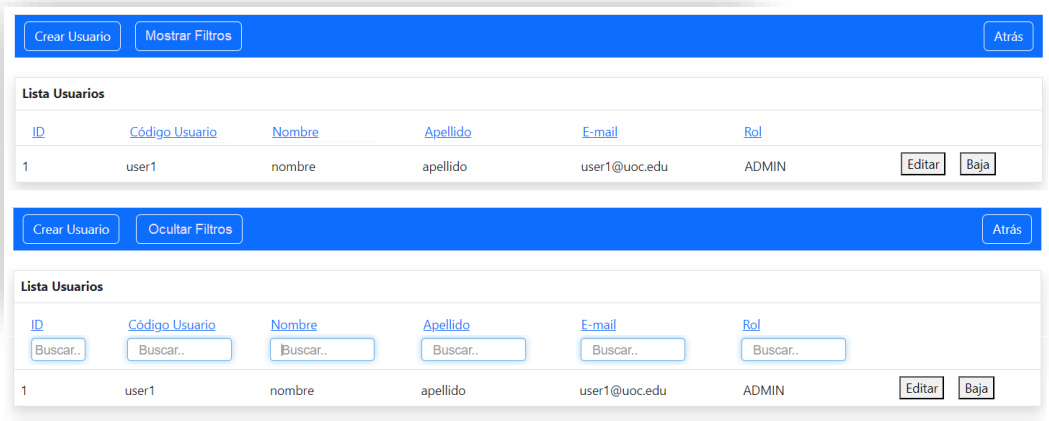

Figura 37 Pantalla de la lista de usuarios

4.3 Sprint Review

En este *Sprint*, la documentación realizada en *Jira* y *Confluence* ha sido la siguiente:

- Se ha incluido la reunión relativa a la *Review* del *Sprint 1*.

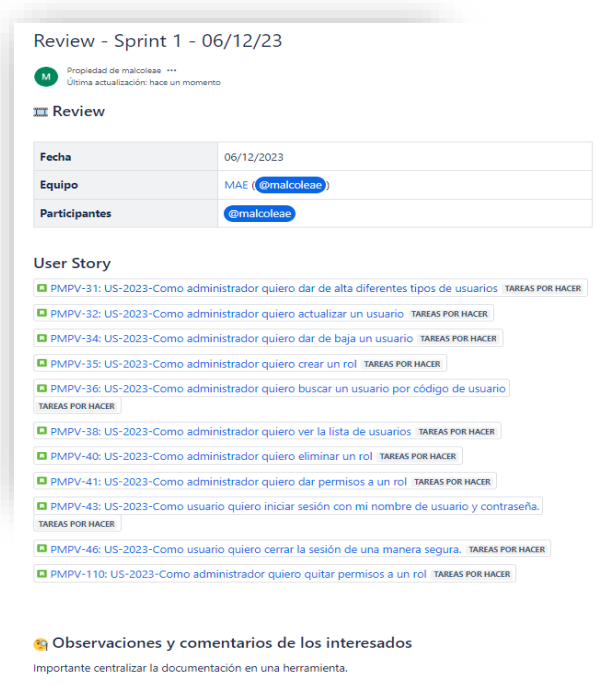

Figura 38 Review – Sprint 1

- El análisis y diseño de la gestión de usuarios teniendo en cuenta las buenas prácticas de CMMI y la norma ISO/IEC/IEEE 12207:2017. De modo que el documento de análisis y diseño [Anexo 3] se ha asociado a todas las historias de usuario que estaban involucradas en ello.

A través del acceso “Páginas de proyectos” en *Jira,* se podrá acceder a la documentación incluida en *Confluence*.

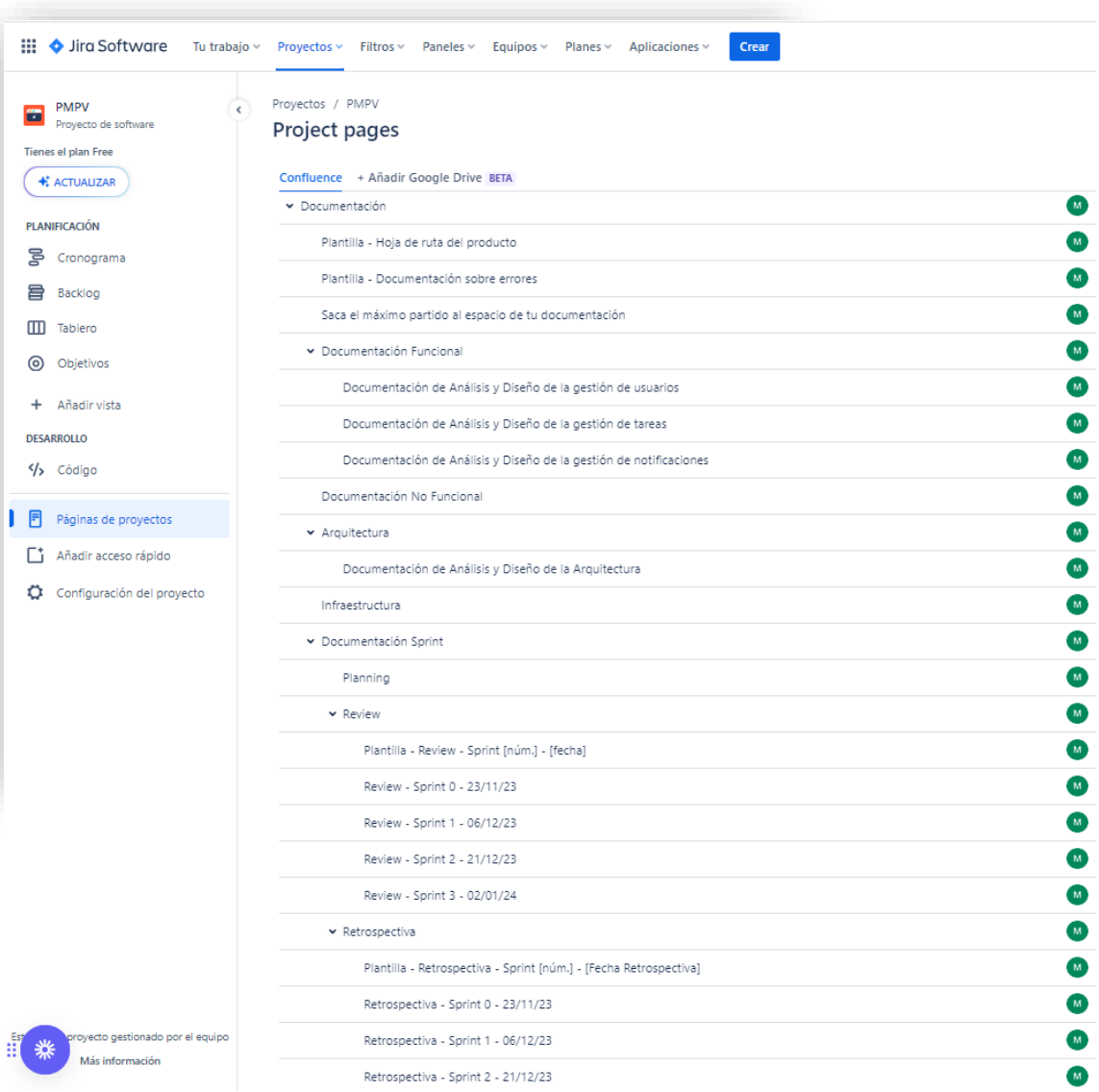

Figura 39 Retrospectiva – Páginas de proyectos

4.4 Sprint Retrospective

Del mismo modo que el primer Sprint, se ha documentado en Miro y Confluence:

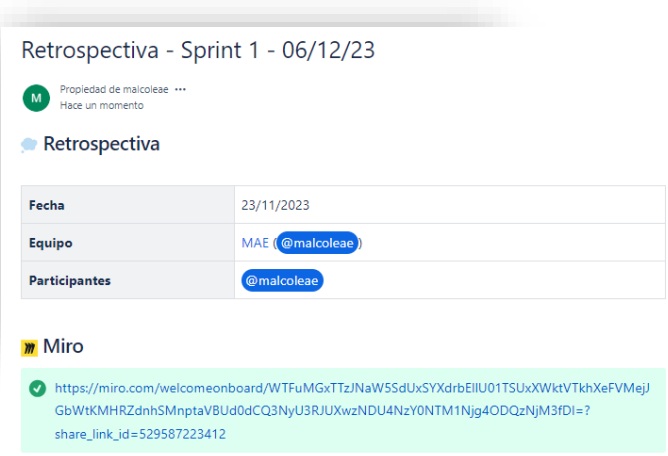

Figura 40 Retrospectiva - Sprint 1 – Documentación Confluence

En este Sprint, se ha puesto de manifiesto lo interesante que ha sido la adquisición de conocimientos relativos a la autenticación y autorización de Spring Boot.

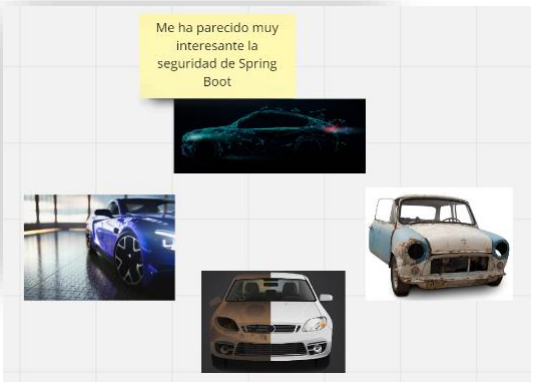

Figura 41 Retrospectiva - Sprint 1 – Miro – Emociones

Respecto a las valoraciones de este Spring, se ha expuesto, la diversidad de diagramas que se han utilizado, la implicación del tutor en las valoraciones realizadas y los problemas que se han tenido a la hora de documentar en *Jira* por dejarlo para el final.

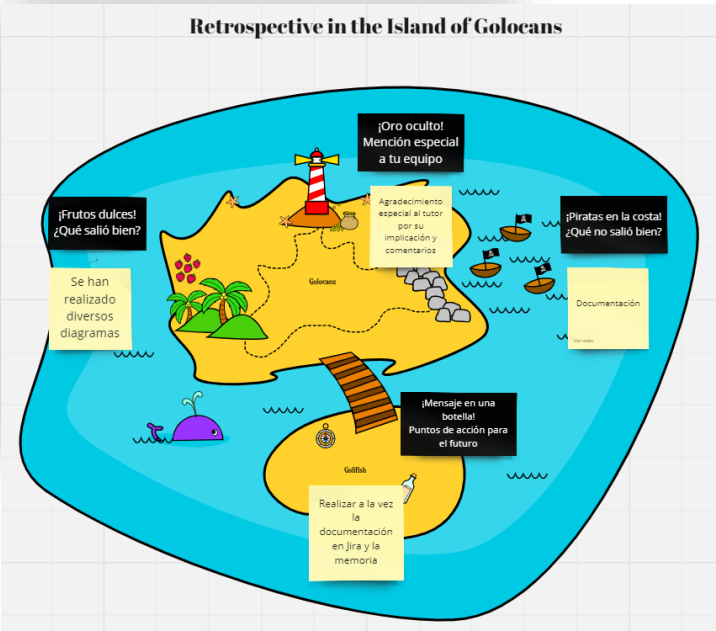

Figura 42 Retrospectiva - Sprint 1 – Miro – Comunicación

Por lo tanto, una de las acciones a tomar es realizar al mismo tiempo la documentación en *Jira* y en la memoria.

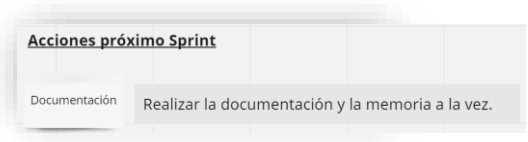

Figura 43 Retrospectiva - Sprint 1 – Miro – Acciones

Finalmente, es importante destacar que se ha mitigado el impacto en las modificaciones de la memoria al recibir *feedback* más a menudo del tutor. Sin embargo, en este *Sprint* se han excedido las horas planificadas en 6 horas, lo cual ha ocasionado un aumento en el presupuesto de 120 euros. Esto ha sido debido al número de historias que se incluyeron en el *Sprint*.

Presupuesto Inicial: 7119 €
Contingencias: **339 €**

Desviación Sprint 0: 180 €
Desviación Sprint 1: 120 €
Desviación Total: **300 €**

Tal y como se puede observar, hasta el momento, la desviación está cubierta por las contingencias y no ha afectado a ninguna entrega.

5. Sprint 2

5.1 Sprint Planning

El objetivo de este *Sprint* es llevar a cabo el análisis y el diseño relativo a la gestión de tareas de mantenimiento, tanto manuales como las proporcionadas por el fabricante. A continuación, se exponen las *User Stories* que se han estimado y acordado realizar a lo largo del *Sprint*:

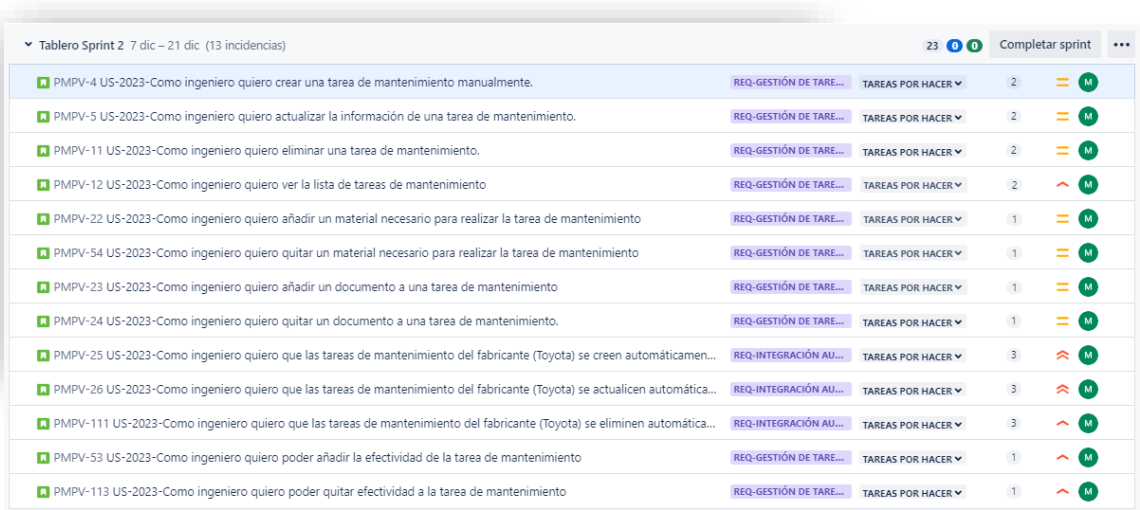

Figura 44 Planificación Sprint 2 en Jira

5.2 Ejecución Sprint

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.

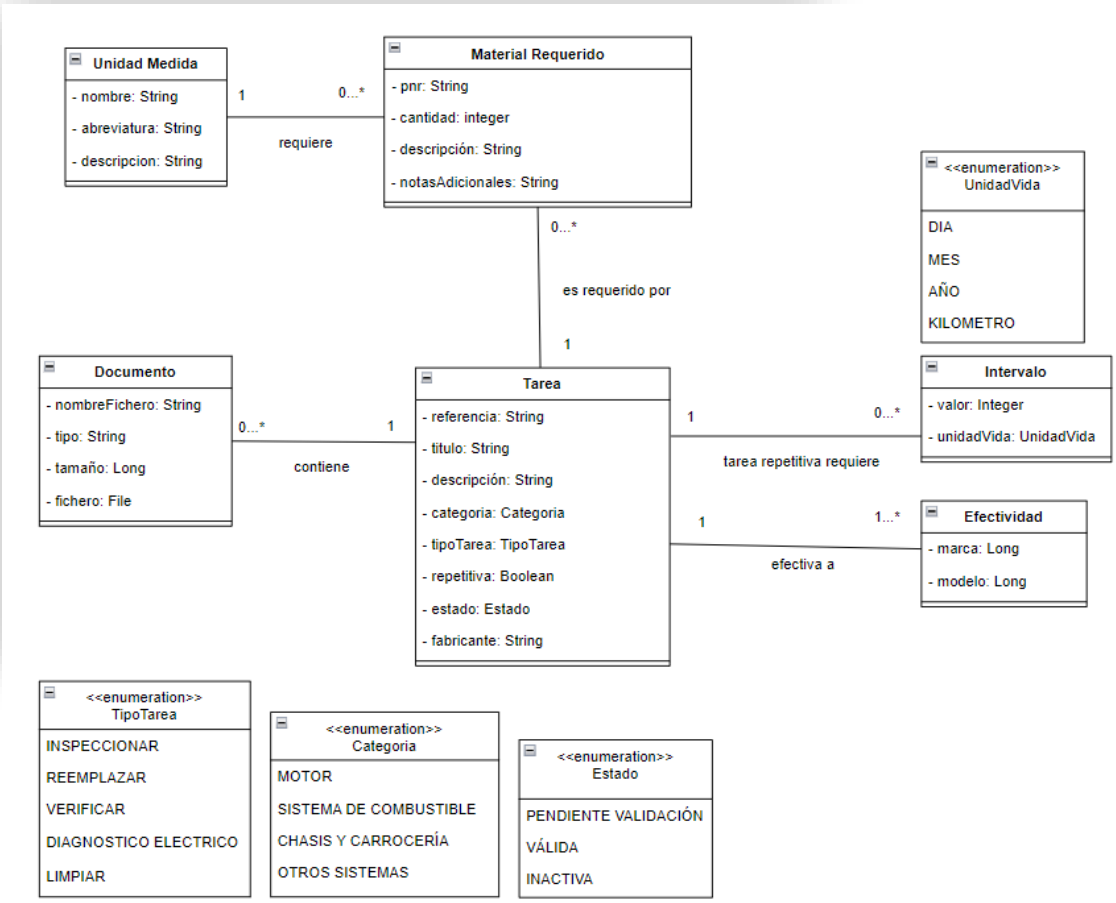

Figura 45 Modelo de dominio del Servicio Tarea

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.

Tabla 13 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.

Tabla 14 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:

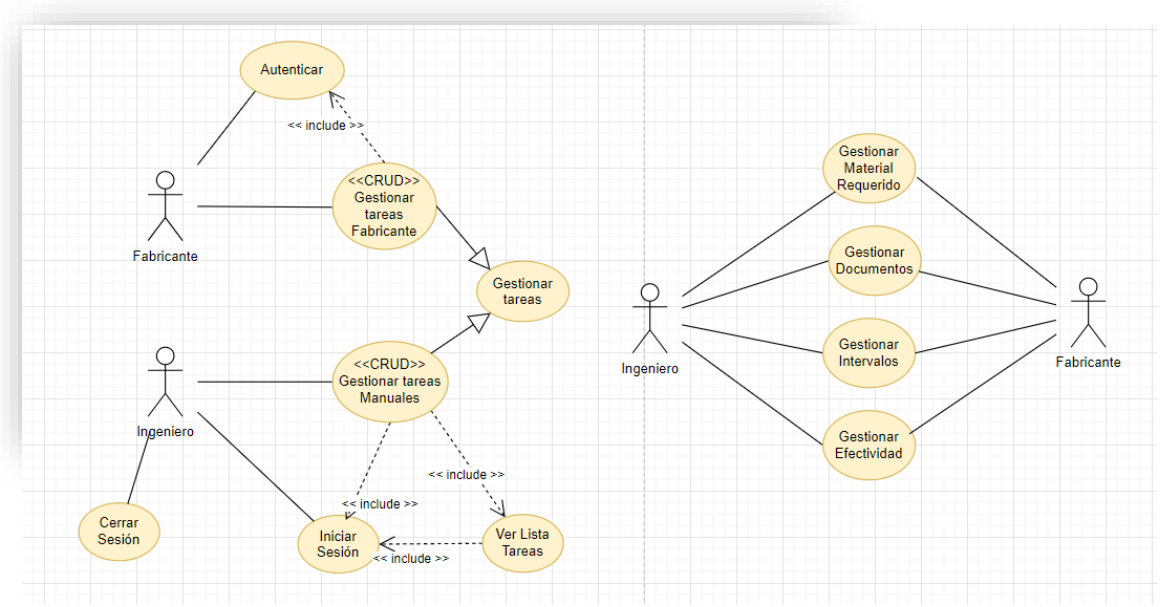

Figura 46 Diagrama de casos de uso de la gestión de tareas

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 de 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.

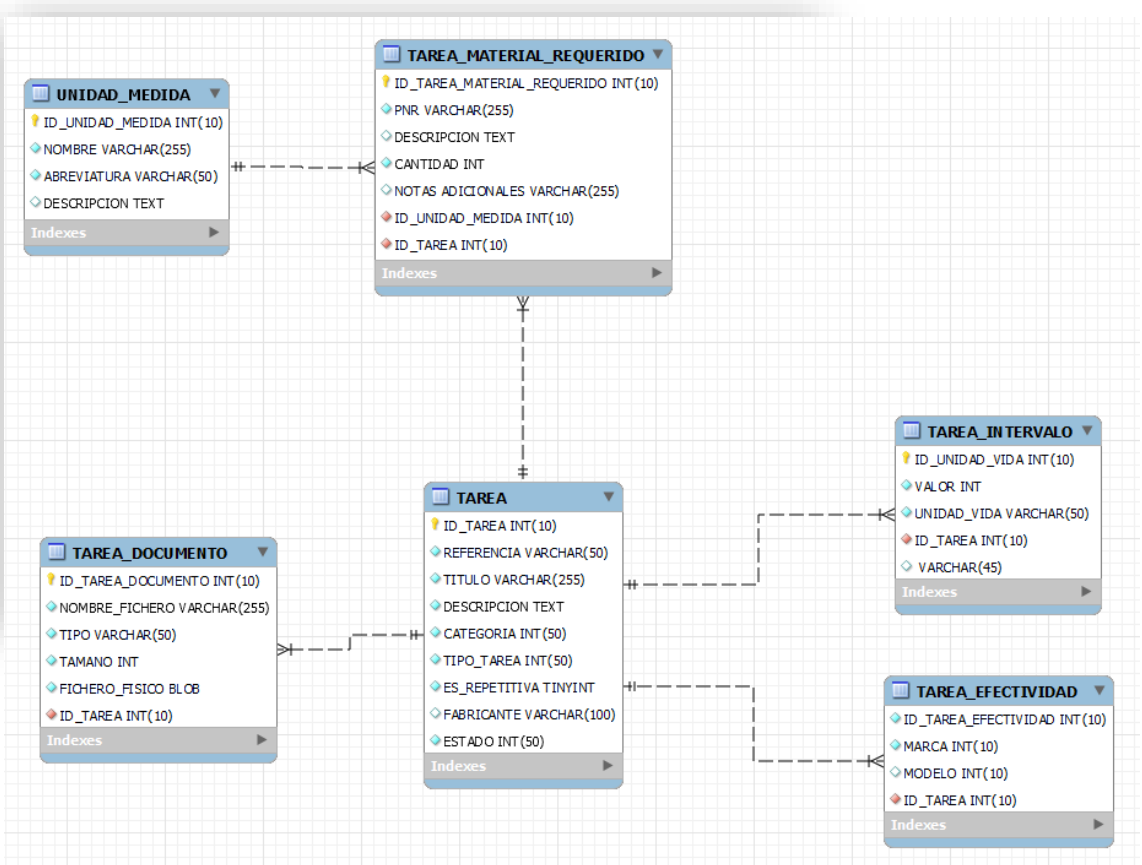

Figura 47 Diagrama relacional del modelo de datos de la gestión de tareas

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.

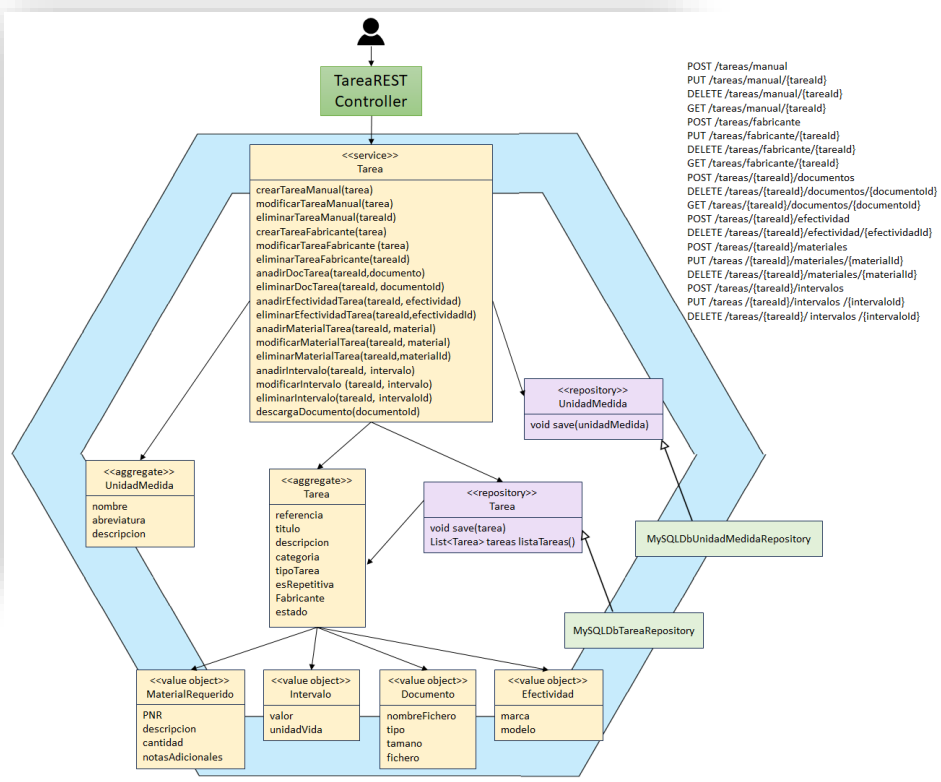

Figura 48 Diseño del Servicio Tarea

Los endpoints definidos en este servicio son los siguientes:

- Para las tareas manuales:

Método: POST
URL: **tareas/manual**

Método: DELETE
URL: **tareas/manual/{tareaId}**

Método: PUT
URL: **tareas/manual/{tareaId}**

Método: GET
URL: **tareas/manual/{tareaId}**

- Para las tareas del fabricante:

  Método: POST
  URL: **tareas/fabricante**

  Método: DELETE
  URL: **tareas/ fabricante/{tareaId}**

  Método: PUT
  URL: **tareas/fabricante/{tareaId}**

  Método: GET
  URL: **tareas/fabricante/{tareaId}**

- 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}**

- Para las efectividades de la tarea:

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

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

- 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}**

- 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.

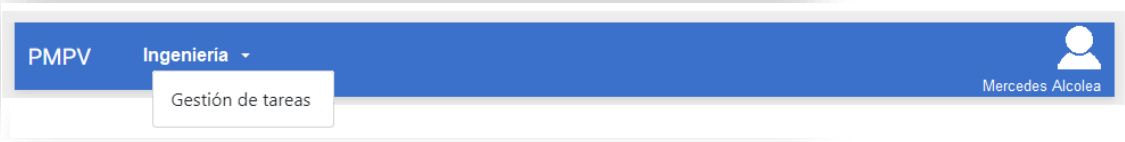

Figura 49 Menú Principal Rol Ingeniero

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.

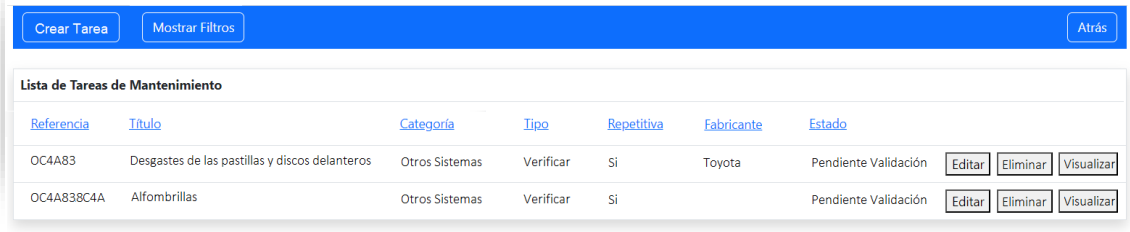

Figura 50 Ventana “Lista de Tareas de Mantenimiento”

Las siguientes ventanas permiten crear y actualizar tareas manuales.

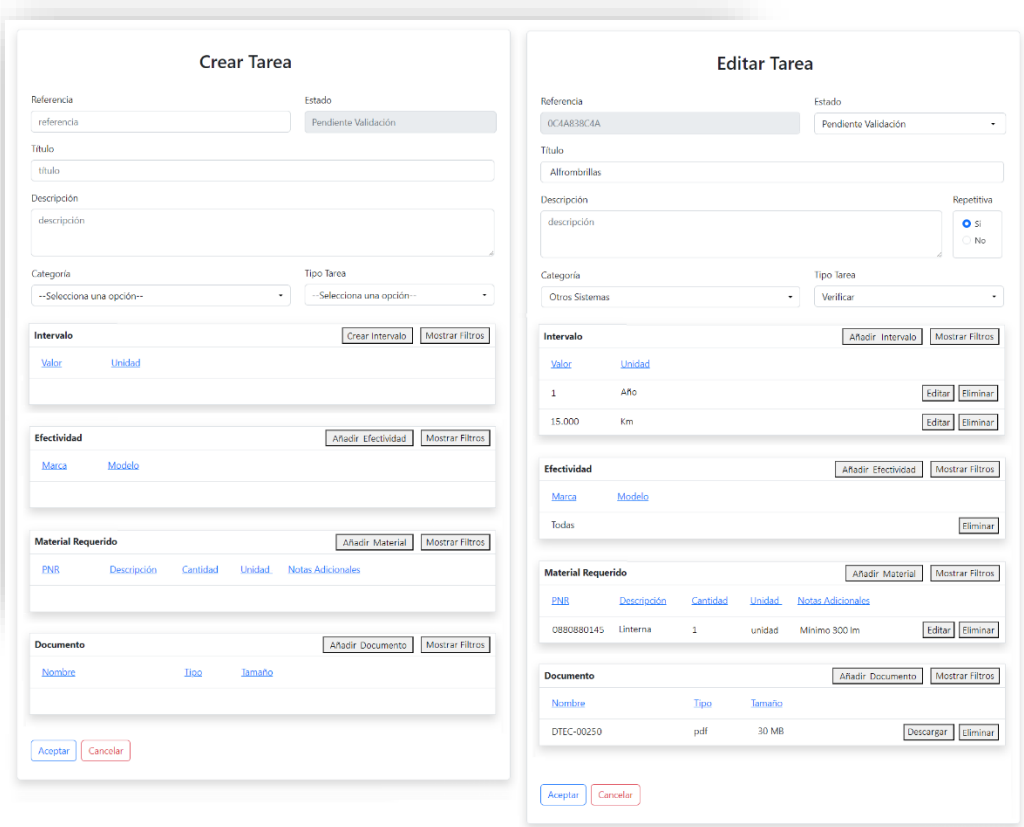

Figura 51 Ventana “Crear Tarea” y “Editar Tarea”

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.

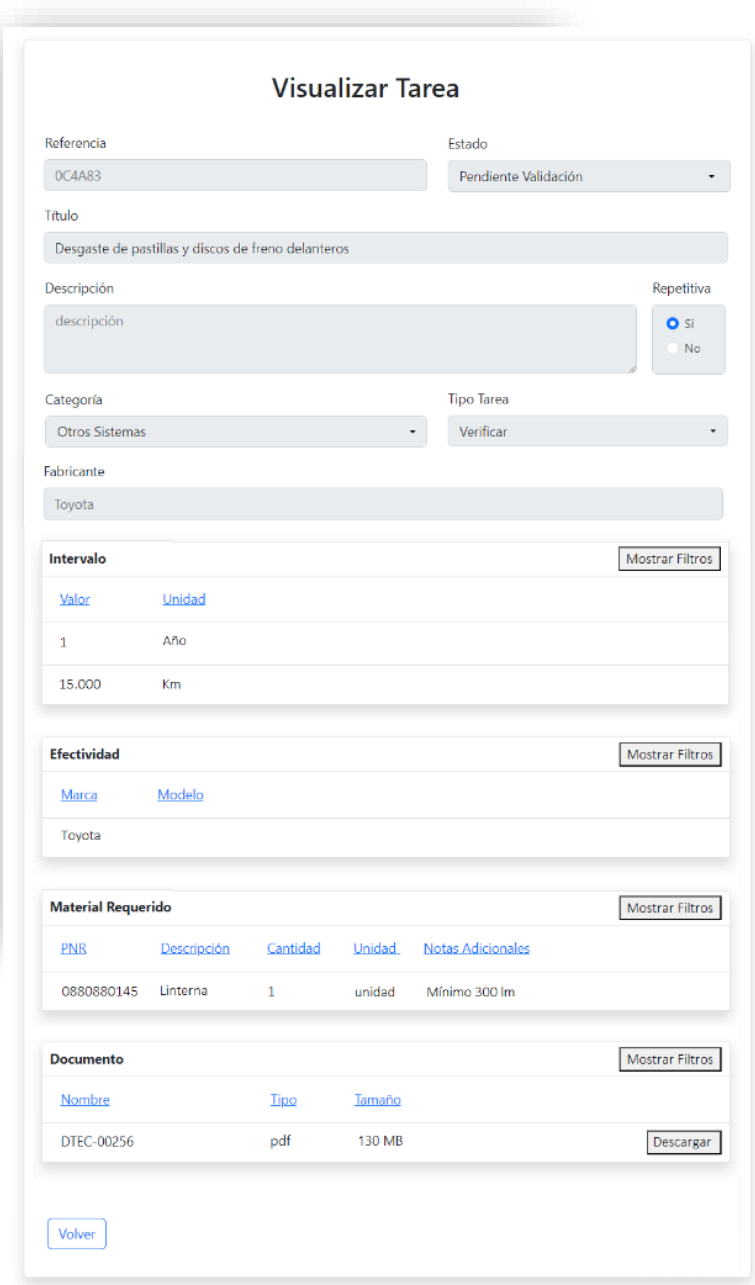

Figura 52 Ventana “Visualizar Tarea”

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*, *Concordion* 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 *Concordion*.

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 *Concordion*.

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 *Concordion* 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.

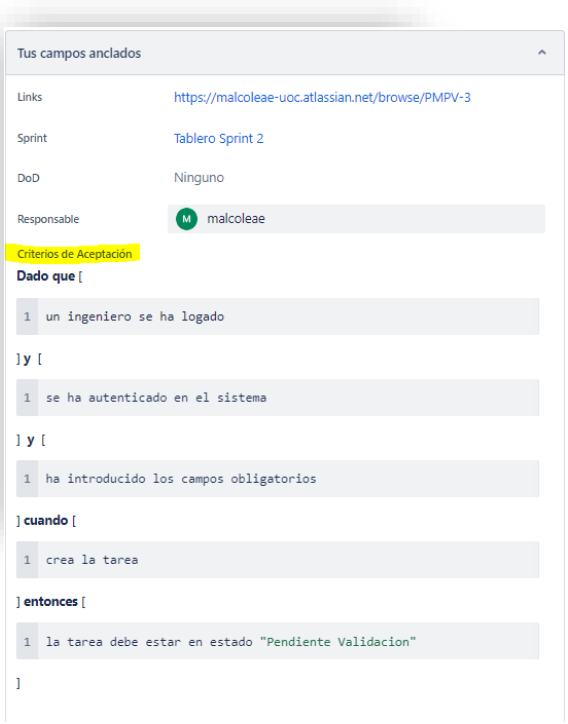

Figura 53 Criterio de aceptación relativo a la creación de una tarea

Una vez definidos los criterios de aceptación con sus correspondientes escenarios, estos son ejecutados de forma automática en *Jenkins*. Para que puedan ser comprendidos tanto por los interesados no técnicos como por el equipo de desarrollo, se escriben en ***Gherkin***[19].

Gherkin permite definir escenarios que describen procedimientos mediante el desarrollo guiado por comportamiento (BDD)[20]. 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.

[19] Gherkin es un lenguaje especifico de dominio (DSL) que en este contexto describe la estructura y contenido de una historia de usuario [35].
[20] BDD o *Behaviour Driven Development* es un conjunto de prácticas destinadas a mejorar la colaboración entre desarrolladores y partes interesadas no técnicas [36][36].

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 Concordion* 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:

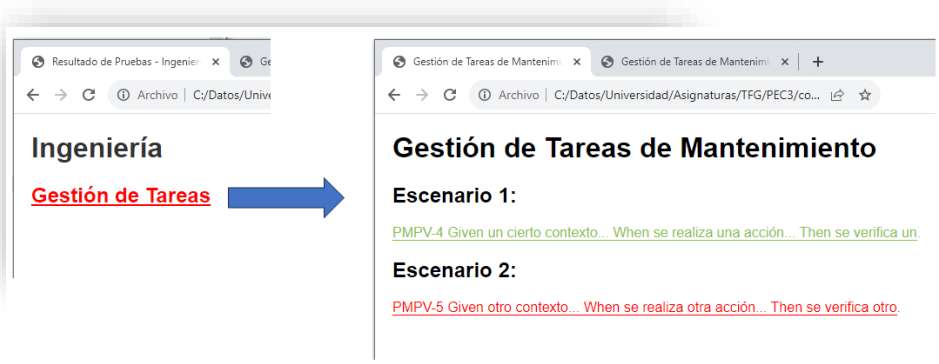

Figura 54 Visualización del resultado de las pruebas de los criterios de aceptación

Además, *Concordion* 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.

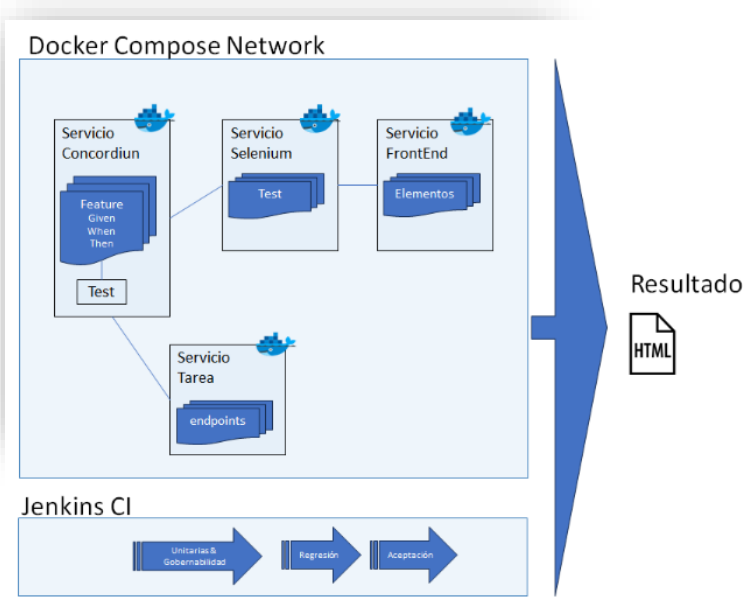

Figura 55 Automatización de Pruebas

5.3 Sprint Review

Tal y como se puede observar a continuación, las historias incluidas en la *Review* del *Sprint* 2 se han documentado en *Jira* y *Confluence.*

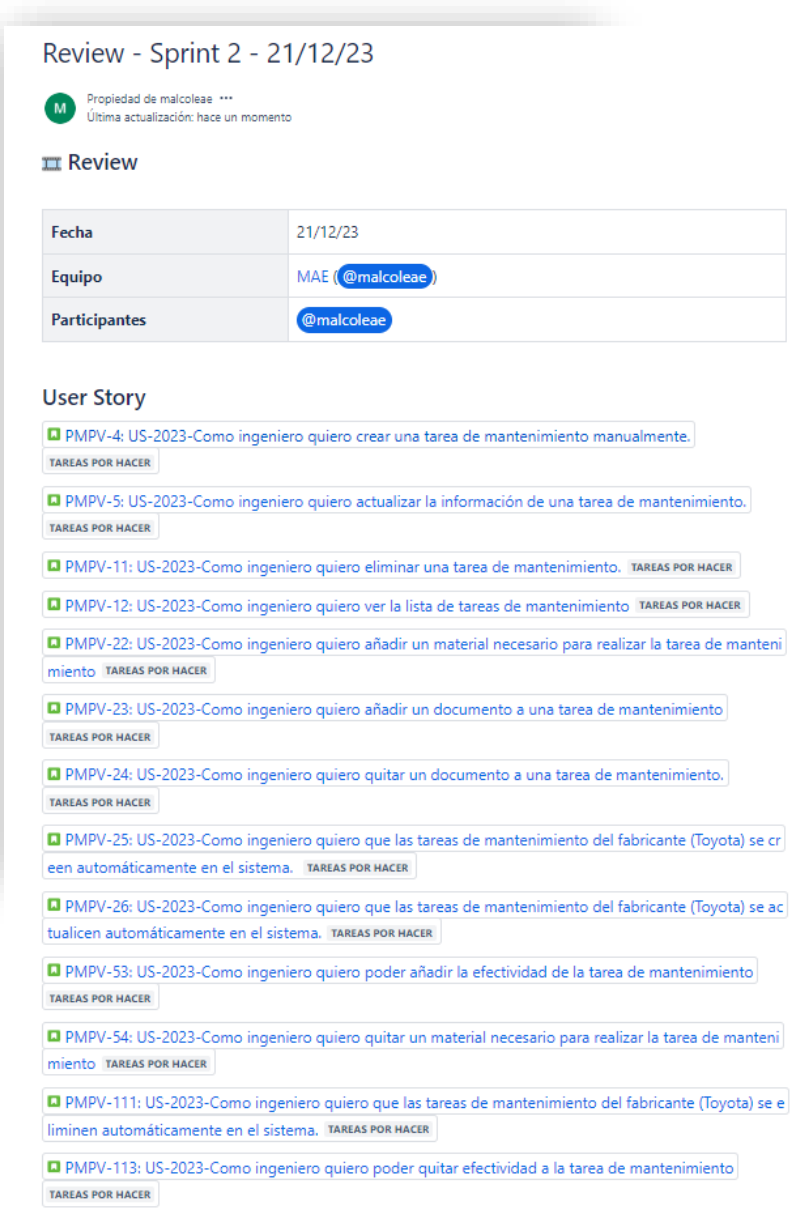

Figura 56 Review – Sprint 2

El documento de análisis y diseño del servicio relativo a las tareas de mantenimiento [Anexo 4] ha tenido en cuenta las buenas prácticas de CMMI y la norma ISO/IEC/IEEE 12207:2017.

5.4 Sprint Retrospective

La retrospectiva se ha documentado en Miro y Confluence:

Figura 57 Retrospectiva - Sprint 2 – Documentación Confluence

En este Sprint, se ha experimentado un sentimiento de satisfacción al tener un progreso constante en las tareas planificadas.

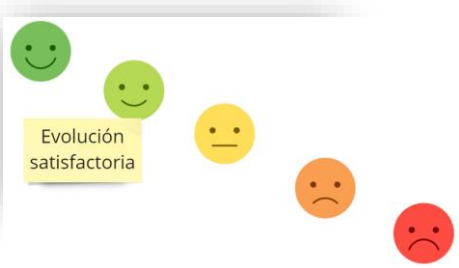

Figura 58 Retrospectiva - Sprint 2 – Miro – Emociones

Respecto a las valoraciones de este Spring, el formato utilizado ha sido relativo a las navidades, donde se ha reflejado el limitado conocimiento que se tenía sobre la automatización de las pruebas. También, como llevar a cabo la documentación ha sido más fácil gracias a la experiencia adquirida en los Sprints anteriores y, finalmente, cuán importante es mantener la constancia en el trabajo diario para lograr alcanzar los hitos establecidos.

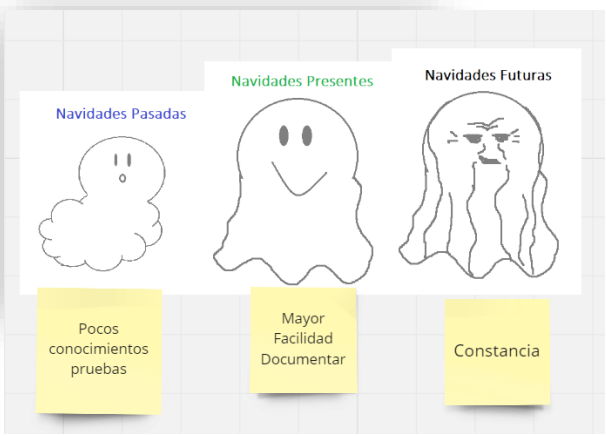

Figura 59 Retrospectiva - Sprint 2 – Miro – Comunicación

Por lo tanto, la acción a tomar para el próximo Sprint es la constancia del trabajo estableciendo un horario concreto.

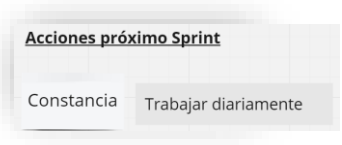

Figura 60 Retrospectiva – Sprint 2 – Miro – Acciones

Finalmente, es importante destacar que en este *Sprint* no se ha tenido ningún tipo de desviación.

6. Sprint 3

6.1 Sprint Planning

El objetivo de este *Sprint* es llevar a cabo el análisis y el diseño relativo a la gestión de notificaciones. A continuación, se exponen las *User Stories* que se han estimado y acordado realizar a lo largo del *Sprint*:

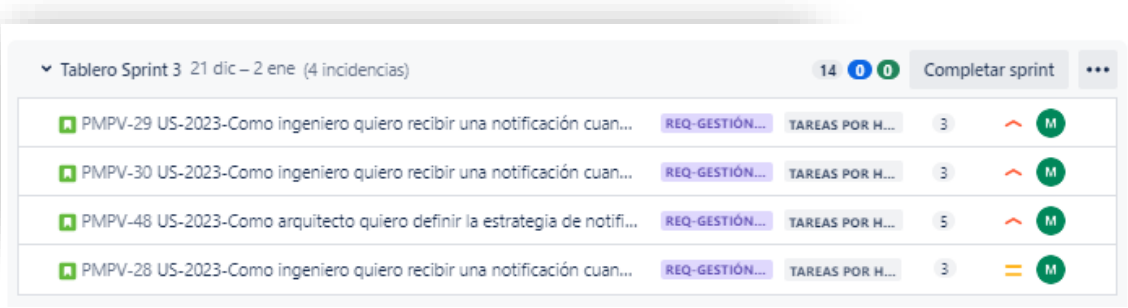

Figura 61 Planificación Sprint 3 en Jira

6.2 Ejecución Sprint

Especificación

Teniendo en cuenta las siguientes historias de usuario acordadas anteriormente:

- US-2023-Como ingeniero quiero recibir una notificación cuando se haya actualizado una tarea de mantenimiento por parte del fabricante.

- US-2023-Como ingeniero quiero recibir una notificación cuando se haya eliminado una tarea de mantenimiento por parte del fabricante que esté asociado a un plan de mantenimiento

- US-2023-Como arquitecto quiero definir la estrategia de notificaciones

- US-2023-Como ingeniero quiero recibir una notificación cuando se haya creado una nueva tarea de mantenimiento por parte del fabricante

El fabricante puede realizar diversas solicitudes a través de los *endpoint* del Servicio de Tareas. Estas solicitudes incluyen la creación de una nueva tarea de mantenimiento, la actualización de los datos de una tarea existente o la eliminación de una tarea que ya no es necesaria. A continuación, se detallan los distintos endpoints del servicio relativos a la gestión de tareas del fabricante:

Creación de una tarea de mantenimiento por parte del fabricante:
- Método: POST
- URL: **tareas/fabricante**

Actualización de una tarea de mantenimiento por parte del fabricante:
- Método: PUT
- URL: **tareas/fabricante/{tareaId}**

Eliminación de una tarea de mantenimiento por parte del fabricante:

- Método: DELETE
- URL: **tareas/fabricante/{tareaId}**

Una vez validada y completada la acción, se genera un evento correspondiente a la creación, actualización o eliminación. Este evento es publicado en el bus de mensajes Kafka. Posteriormente, el Servicio de Notificación se conecta, recibe el mensaje, obtiene las direcciones de correo electrónico de los ingenieros y les envía un correo electrónico informándoles de dichas acciones.

Colaboraciones entre servicios

A partir de la especificación, para comprender mejor la relación entre los diversos componentes e identificar las colaboraciones entre los diferentes servicios y dependencias, se definen los servicios, las operaciones y las colaboraciones que se ven afectados durante el proceso indicado anteriormente.

Servicio	Operación	Colaboración
Servicio Tarea	crearTareaFabricante() modificarTareaFabricante() eliminarTareaFabricante() obtenerDetalleTarea()	
Servicio Notificación	enviarCreacionTarea() enviarModificacionTarea() enviarEliminacionTarea()	▪ Servicio Usuario: obtenerCorreoIngenieros() ▪ Servicio Tarea: obtenerDetalleTarea()
Servicio Usuario	obtenerCorreoIngenieros()	

Tabla 15 Servicios Operaciones Colaboraciones

En este proceso se ven involucrados tres servicios, donde:

- El **servicio de tarea** gestiona las tareas de mantenimiento manuales y del fabricante y cuyas operaciones:
    - *crearTareaFabricante()*: Permite al fabricante crear una nueva tarea de mantenimiento.
    - *modificarTareaFabricante()*: Permite al fabricante actualizar una tarea existente del fabricante.
    - *eliminarTareaFabricante()*: Permite al fabricante eliminar una tarea existente del fabricante.
    - *obtenerDetalleTarea()*: Permite obtener el detalle de una tarea en concreto.
- El **servicio de usuario** gestiona los usuarios y cuya operación:

- *obtenerCorreoIngenieros()*: Permite recuperar las direcciones de correo electrónico de los usuarios que tienen el rol de ingeniero.

- El **servicio de notificaciones** gestiona las notificaciones entre los diferentes servicios y cuyas operaciones:

  - *enviarCreacionTarea()*: Se encarga de enviar notificaciones cuando se crea una nueva tarea de mantenimiento.

  - *enviarModificacionTarea()*: Se encarga de enviar notificaciones cuando se actualiza una tarea existente.

  - *enviarEliminacionTarea()*: Se encarga de enviar notificaciones cuando se elimina una tarea.

- El **Servicio Tarea** publica eventos de creación, modificación y eliminación de tareas en el bus de mensajes Kafka, y el **Servicio Notificación** consume estos eventos para enviar correos electrónicos a los ingenieros.

- El **Servicio Notificación** colabora con el **Servicio Usuario** para obtener las direcciones de correo electrónico de los ingenieros a quienes se enviarán las notificaciones.

Diseño notificaciones

A continuación, se establece el diagrama que muestra la interacción entre los servicios de tarea, notificación y usuario en una arquitectura distribuida que permite visualizar de una forma fácil como se comunican entre sí.

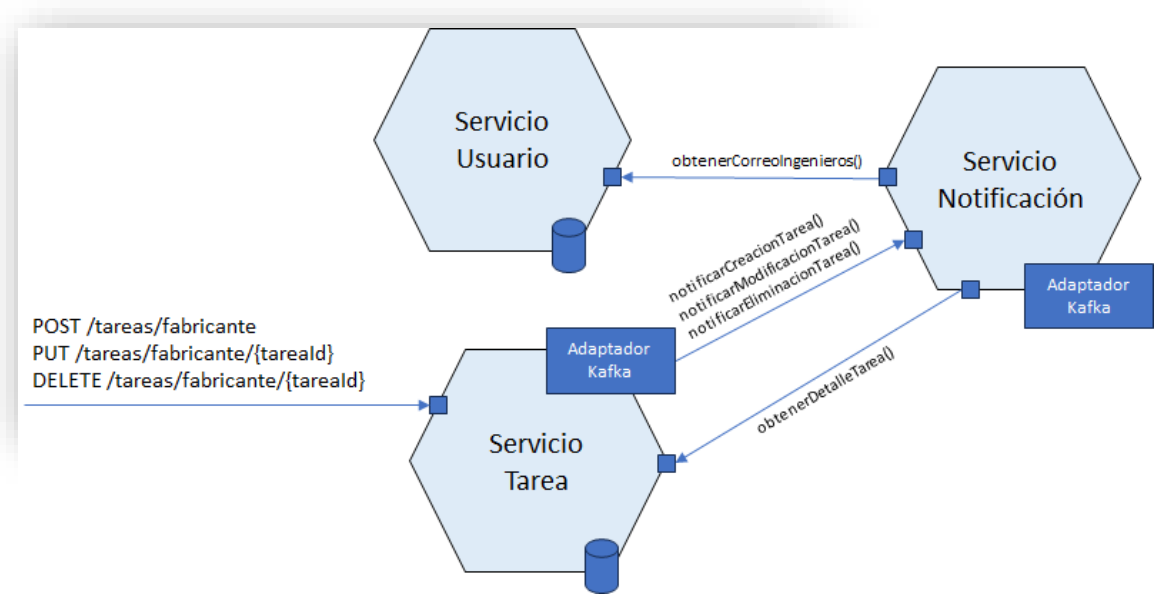

Figura 62 Diagrama de interacción entre servicios

Tal y como se puede observar en el diagrama, los servicios de tarea y notificación deberán tener un adaptador para poder recibir y enviar notificaciones. Sin embargo, la obtención de información entre los diferentes servicios se realizará

a través de una comunicación vía REST. A continuación, se explica en detalle la estrategia que se llevará a cabo para realizar dichas notificaciones.

Estrategia de Notificaciones

La estrategia de notificaciones debe tener en cuenta los siguientes requisitos no funcionales:

- **Escalabilidad**. Permite manejar grandes volúmenes de mensajes.

- **Distribución Efectiva**. Distribución que facilita la comunicación entre los diferentes servicios.

- **Tolerancia a fallos**. Ofrece una solución a los fallos que se puedan producir asegurando la continuidad del proceso.

Para establecer una comunicación efectiva y atómica entre los servicios de tareas, notificaciones y usuarios, y garantizar que la arquitectura cumpla con las características indicadas anteriormente, se aplica el patrón mensajería y el patrón bandeja de salida transaccional.

El **patrón mensajería** permite la comunicación entre servicios de forma asíncrona, es decir, mediante el intercambio de mensajes a través de canales de mensajería sin necesidad de que ambos servicios estén disponibles. Para ello, se utiliza el patrón Publicador/Suscriptor que facilita que un servicio publique un mensaje para que otro u otros consumidores lo puedan recibir.

Para que el proceso sea atómico y se eviten problemas de inconsistencias de datos como, por ejemplo, cuando se crea, actualiza o elimina una tarea y al producirse un fallo de base de datos el mensaje se envía, se utiliza el **patrón bandeja de salida transaccional** o eventos de aplicación.

El patrón bandeja de salida transaccional publica un mensaje como parte de una transacción de base de datos, es decir, se envía el mensaje si se confirma la transacción en base de datos, en caso contrario no se envía. Además, los eventos se enviarán en el mismo orden en el que fueron realizados.

Por lo tanto, el servicio que envía el mensaje almacena el evento en la base de datos como parte de la transacción y luego, un proceso intermedio lo envía al bus de mensajería tal y como se puede observar en el siguiente diagrama:

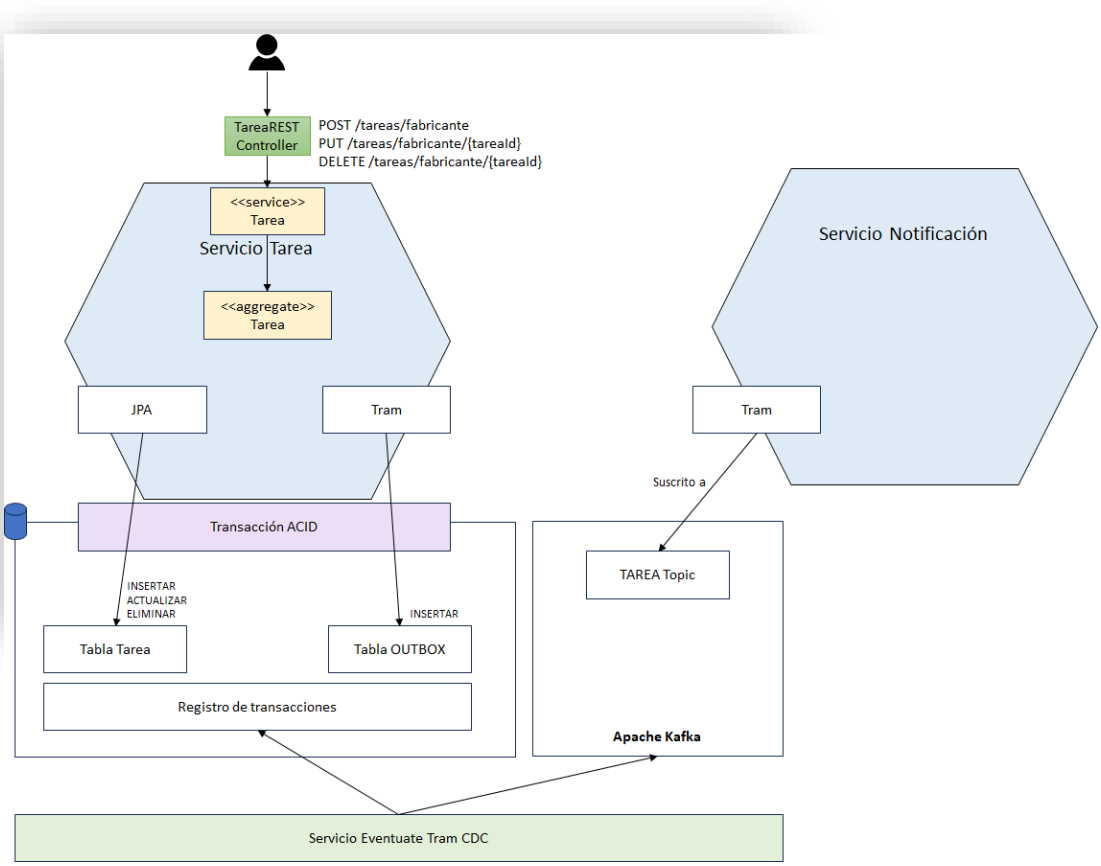

Figura 63 Diagrama del Patrón Bandeja de Salida Transaccional [38]

Para poder llevar a cabo lo indicado anteriormente, se utilizarán las siguientes tecnologías las cuales se deberán integrar con Spring Boot:

Apache Kafka se utiliza como un sistema de mensajería para el intercambio de eventos entre diferentes servicios de una aplicación distribuida. Además, permite gestionar flujos masivos de datos de una forma eficaz. Algunas de sus características principales son:

- Permite la publicación y subscripción de eventos en tiempo real gracias al patrón Publicación/Subscripción.

- Los eventos se almacenan permitiendo su reprocesamiento.

- Es altamente escalable y permite manejar grandes volúmenes de eventos.

- Garantiza la replicación de los eventos entre los diferentes nodos del clúster.

Apache Zookeeper se emplea para coordinar y gestionar la configuración de los nodos del clúster de Kafka facilitando su gestión y mejorando su eficacia.

A continuación, se muestra como Apache Kafka y Zookeeper llevan a cabo la transmisión de mensajes.

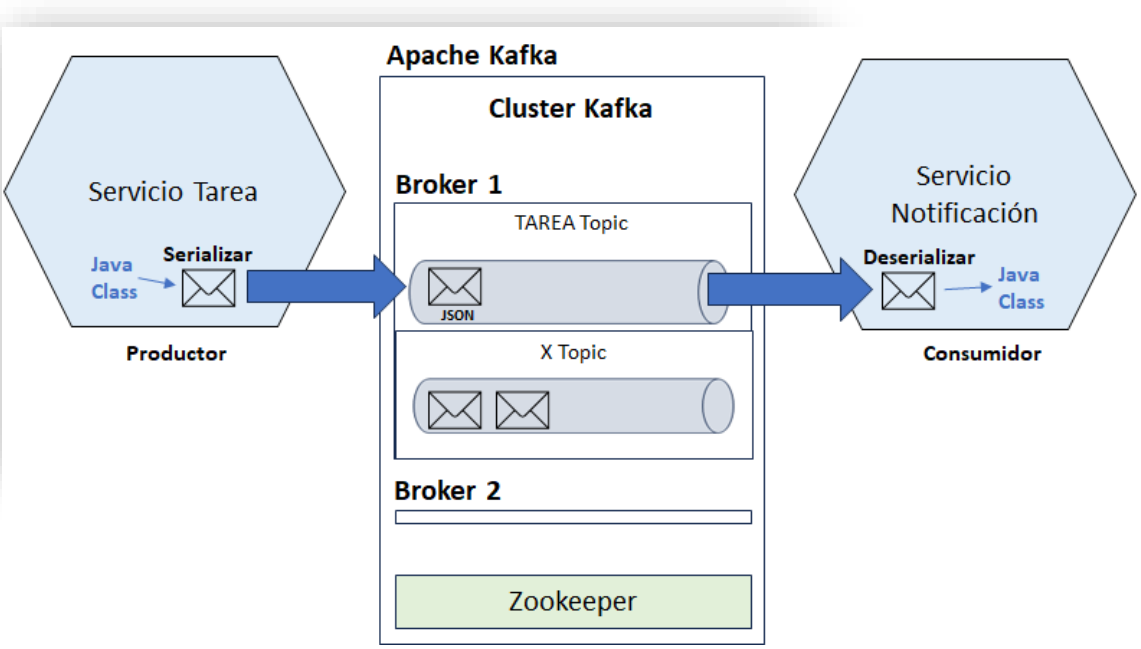

Figura 64 Diagrama de comunicación de mensajes del Servicio Tarea al Servicio Notificación

Para que el Servicio Tarea actúe como un **publicador** o **productor** de Kafka e interactúe de forma efectiva con Kafka, se tiene que configurar la siguiente infraestructura:

KafkaProducerConfig. Una clase de configuración que permite el envío de mensajes utilizando Apache Kafka. Además, debe tener los siguientes métodos:

- **ProducerFactory**(): Crea y devuelve una instancia de ProducerFactory, el cual es responsable de producir instancias de productores de Kafka. Esta clase incluye la configuración necesaria para establecer la conexión con los servidores de Kafka, como la dirección del bus de mensajes de Kafka, la clave con la que se serializan los mensajes y el serializador para convertir los objetos Java en formato JSON.

- **KafkaTemplate**(): Crea y devuelve un objeto KafkaTemplate que utiliza la configuración de ProducerFactory. Este objeto proporciona el método enviar que permite enviar mensajes al bus de mensajes de Kafka.

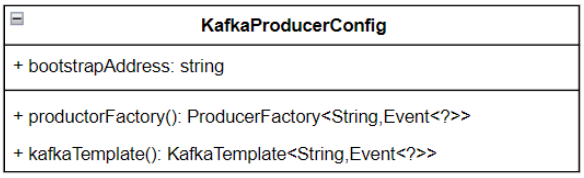

Figura 65 Configuración Productor Kafka

Tal y como se puede observar, en este caso, se puede enviar cualquier mensaje que cumpla la clase abstracta *Event* en la cual se informa un identificador, una

fecha, el tipo de acción que es (creación, actualización o eliminación) y la propia tarea.

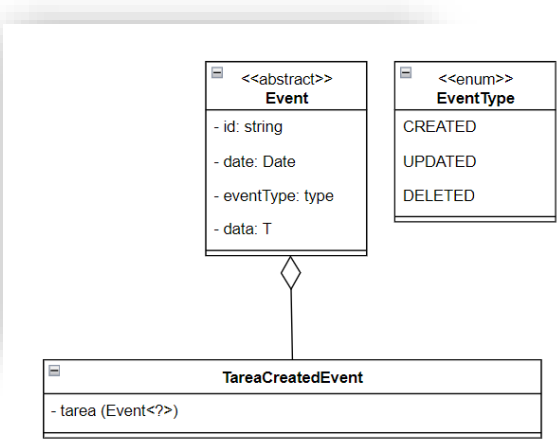

Figura 66 Evento que se envía a otro servicio

Una vez configurado, es necesario inyectar el objeto KafkaTemplate en el lugar donde se quiere enviar el evento, en este caso, se crea una clase que contine el método que lo publicará y que será llamado desde el servicio de la tarea cuando se cree, actualice o elimine una tarea de fabricante.

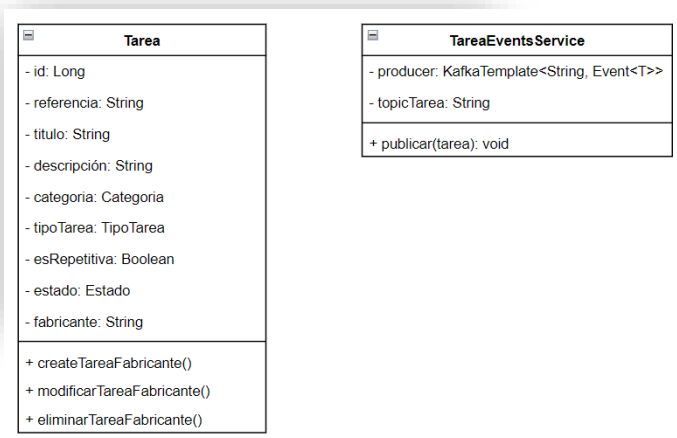

Figura 67 Servicio que publica el evento

Para enviarlo a través del método *SEND* de KafkaTemplate se debe especificar el topic al que se enviará y el evento que se enviará. Este último, extiende de la clase Event donde se incluye el tipo de acción (creación, actualización o eliminación), el identificador del evento, la fecha de creación y la tarea.

Por lo tanto, con el topic y los datos de configuración del productor como es la dirección de bus de mensajería, se produce el envío del evento a la correspondiente “tubería” o topic de Kafka en la cual se publica. Es importante resaltar que existen herramientas que permiten ver los mensajes que se han enviado en los diferentes topics de Kafka como, por ejemplo, Offset Explorer.

Finalmente, los consumidores subscritos al topic podrán leer los mensajes. Pero, para ello, es necesario que el servicio Notificación sea configurado como consumidor y, por lo tanto, debe tener la siguiente configuración:

KafkaConsumerConfig. Una clase de configuración que permite conectarse y leer los mensajes de Apache Kafka. Esta clase tiene los siguientes métodos:

- **ConsumerFactory**(): Crea y devuelve una instancia de ConsumerFactory, el cual es responsable de producir instancias de consumidores de Kafka. Esta clase incluye la configuración necesaria para establecer la conexión con Kafka como es la dirección del bus de mensajería. Además, de proporcionar, el deserializador de la clave y del evento que permitirá transformar el JSON en una clase de Java.

- **KafkaListenerContainerFactory**(): Crea y configura un objeto ConcurrentKafkaListenerContainerFactory que crea contenedores de escucha y que utiliza la configuración de ConsumerFactory.

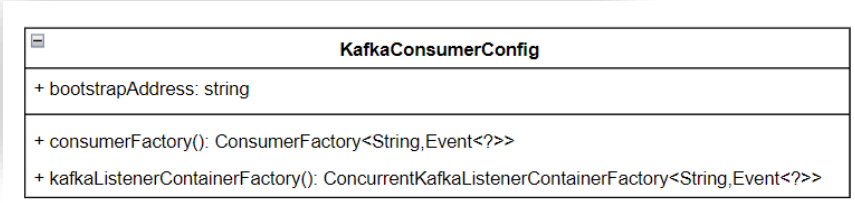

Figura 68 Configuración Consumidor Kafka

En resumen, se establece la configuración necesaria para consumir mensajes desde un servidor Kafka y transformar los mensajes JSON en objetos tipo Event.

Finalmente, para que se ejecute el método que lea los eventos que llegan a un topic de Kafka es necesario configurar lo siguiente:

TareaEventsService: Clase que contiene el método **consumidor**, el cual, es marcado como consumidor de Kafka a través de la anotación **@KafkaListener**[21] donde se especifican los siguientes parámetros:
- El topic sobre el que escucha
- El contenedor de Kafka que se usa para leer el mensaje, en este caso KafkaListenerContainerFactory.
- El grupo de consumidores que lo escuchan.

Todo ello, permite que el método se ejecute cada vez que llegue un mensaje al topic especificado, lo lea y envíe los correos electrónicos a todos aquellos usuarios cuyo perfil sea ingeniero.

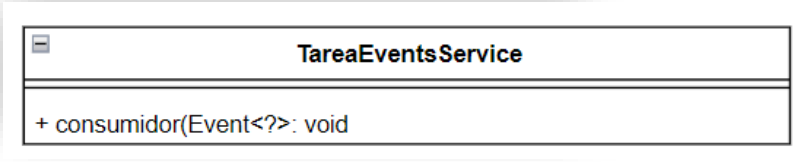

Figura 69 Servicio subscrito al topic de Kafka

[21] Dado que este Trabajo solo contempla el diseño, se debería haber indicado la necesidad de tener un ***listener***, pero dado que se ha especificado la tecnología puede ser interesante conocer que su implementación requerirá la anotación *@KafkaListener*.

A continuación, se muestra el diagrama de actividad donde se puede ver el flujo completo del proceso.

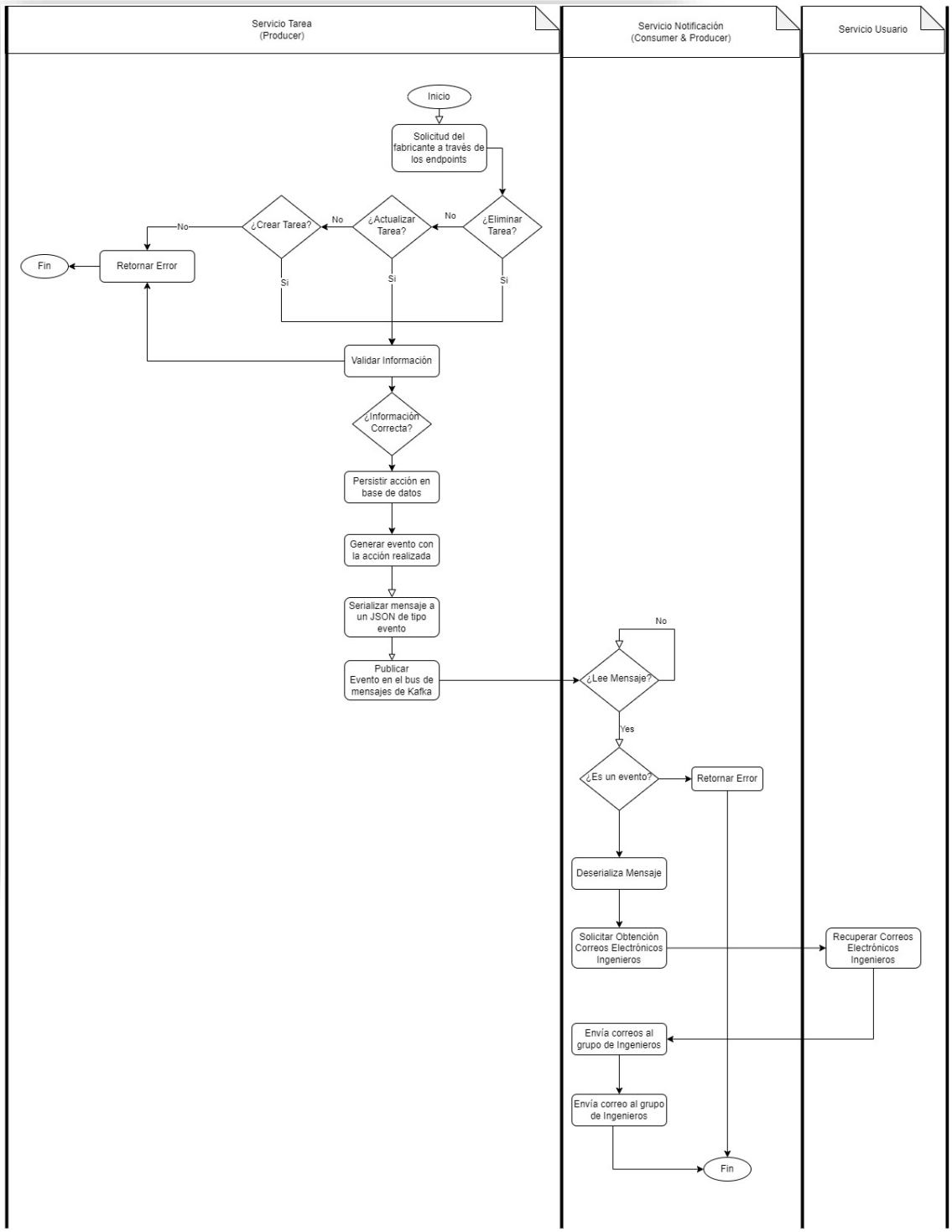

Figura 70 Diagrama de actividad de la creación de tareas del fabricante

Es importante resaltar que la decisión de este estilo arquitectónico ha sido derivada por la adaptabilidad que tiene a los cambios, es decir, la notificación no contiene la información de la tarea o de los usuarios, sino que el servicio notificación tiene que obtenerlo del correspondiente servicio. De esta forma,

existe un mayor acoplamiento entre servicios y aumentaría la gestión de contratos, pero, sin embargo, se adaptaría mejor a posibles cambios que son muy probables dado que estamos en una fase inicial de la aplicación.

6.3 Sprint Review

Del mismo modo que se ha realizado en los Sprints anteriores, la *Review* del *Sprint* 3 se ha documentado en *Jira* y *Confluence* de modo que las historias han sido trazadas con la Review en la cual se han presentado.

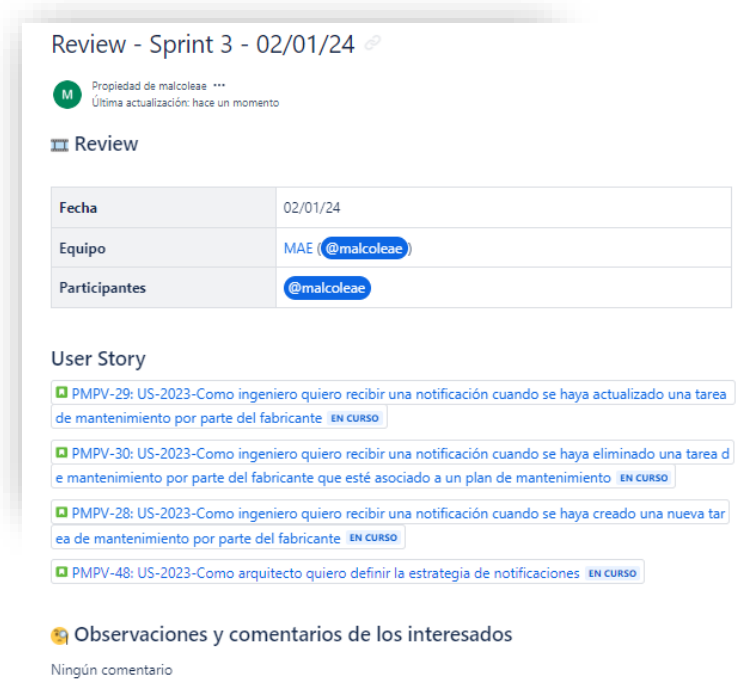

Figura 71 Review – Sprint 3

El documento de análisis y diseño relativo a la gestión de notificaciones[Anexo 4] [Anexo 5] también ha sido documentado tenido en cuenta las buenas prácticas de CMMI y la norma ISO/IEC/IEEE 12207:2017.

6.4 Sprint Retrospective

También, se ha documentado en Confluence la retrospectiva, en la cual se incluye la dirección URL del tablero generado en Miro donde se ha llevado a cabo la actividad.

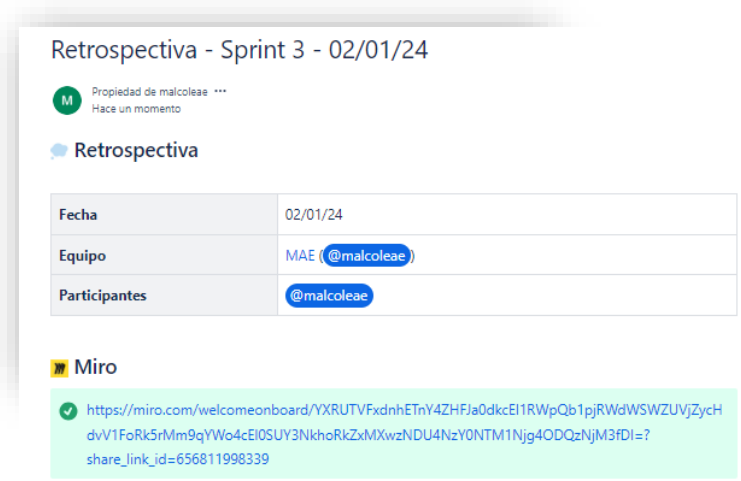

Figura 72 Retrospectiva - Sprint 3 – Documentación Confluence

En este Sprint, se ha experimentado un sentimiento de frustración por la complejidad del tema de notificaciones y por no tener suficiente tiempo para estar con la familia en unas fechas tan señaladas como es la navidad.

Figura 73 Retrospectiva - Sprint 3 – Miro – Emociones [41]

Respecto a las valoraciones de este Spring, se ha destacado los objetivos que se han cumplido, se ha agradecido la rápida respuesta que ha tenido el tutor respecto a las valoraciones que se han solicitado. También, se ha puesto de manifiesto la complejidad del proceso de notificaciones y, finalmente, se ha resaltado como objetivo para otros años intentar realizar entregas antes de fechas señaladas.

Figura 74 Retrospectiva - Sprint 3 – Miro – Comunicación

Por lo tanto, la acción a tomar para próximos *Sprints* es tener en cuenta fechas señaladas de modo que se realicen con anterioridad o pasados unos días.

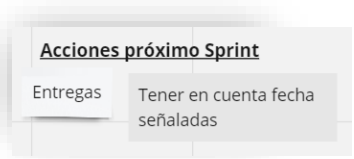

Figura 75 Retrospectiva – Sprint 3 – Miro – Acciones

Finalmente, al igual que en el anterior Sprint, no se ha producido ningún retraso, pero se han reducido 3 horas respecto a las planificadas para este *Sprint*. Por lo tanto, el gasto total del proyecto es el siguiente y estaría cubierto por las contingencias:

Presupuesto Inicial: **7119 €** ➔ Total Gastos: 7119 + 240 = **7359 €**
Contingencias: **339 €**

Desviación Sprint 0: 180 €
Desviación Sprint 1: 120 €
Reducción Horas Sprint 3: 60 €
Desviación Total: **240 €**

7. Conclusiones

Durante la realización de este proyecto se han aprendido diversas lecciones. En primer lugar, la implementación de SCRUM siguiendo la norma ISO/IEC/IEEE 12207:2017 con un nivel de madurez 3 resultó ser un desafío debido a la amplitud y abstracción del marco teórico. Por lo tanto, la experiencia o el asesoramiento son cruciales para discernir si la documentación que se llevará a cabo cubre las necesidades del proyecto y los estándares.

Asimismo, la toma de requisitos es un elemento crítico en cualquier proyecto, ya que conforma sus cimientos. Por lo tanto, mantener una comunicación constante con los interesados, exponer documentos funcionales previos a su desarrollo, presentar prototipos de interfaz y reducir el tiempo de los Sprints a 2 semanas permite simplificar y acotar las historias de usuario. Esto no solo facilita su documentación, compresión e implementación, sino que también contribuye a identificar enfoques erróneos de manera temprana, evitando costos en tiempo y dinero.

De igual modo, establecer un marco de trabajo sólido y realista, estandarizando la documentación y definiendo patrones de diseño claros, proporciona una base sólida al proyecto, especialmente en contextos donde los miembros del equipo experimentan cambios constantes. Por consiguiente, la automatización integral del proceso, desde la construcción hasta el despliegue de la aplicación, con énfasis en la automatización de pruebas, especialmente en los criterios de aceptación debido a su fácil compresión, resulta crucial. De este modo, aunque esto implique una gran inversión inicial, con el tiempo, los beneficios se incrementarían, tanto a nivel de costos como de eficacia, al facilitar la incorporación de nuevas pruebas y la detección precoz de errores.

Respecto a los objetivos planteados inicialmente, se ha logrado establecer un marco de trabajo que permite la integración efectiva de prácticas ágiles, como SCRUM, y estándares de calidad, como la norma ISO/IEC/IEEE 12207:2017 con un nivel de madurez 3, en un sistema de software encargado de la gestión del mantenimiento preventivo vehicular. Este marco de trabajo se ha aplicado de manera efectiva, llevando a cabo la documentación en Jira y Confluence. Durante este proceso, se ha realizado tanto el análisis como el diseño de las siguientes funcionalidades: gestión de usuarios con diferentes roles, autenticación y autorización del sistema, gestión de tareas de mantenimiento, automatización de la generación de tareas a partir de los datos del fabricante y su correspondiente notificación a los ingenieros.

La metodología SCRUM ha ayudado en el seguimiento de la planificación y gracias a las Review y Retrospectivas se han ido viendo puntos a mejorar y cosas que han ido funcionando bien. Se ha ampliado más

contenido ya que la automatización de las tareas por parte del fabricante no aportaba mucho valor incluyendo la automatización de pruebas.

En cuanto a futuras líneas de trabajo, dado que el proyecto se ha centrado en el análisis y diseño del área de ingeniería, se abren diversas posibilidades de desarrollo. En primer lugar, es esencial llevar a cabo la implementación de los desarrollos planificados, ya que, aunque se ha establecido el diseño, aún no se ha implementado ninguna funcionalidad. Posteriormente, se podrían abordar otras áreas de la aplicación, como el mantenimiento y la administración de vehículos, incluyendo la planificación del plan de mantenimiento, la gestión de vehículos y la ejecución de órdenes de mantenimiento. Además, la estandarización del proceso podría ampliarse a todas las fases del proyecto, no limitándose únicamente al ciclo de vida del desarrollo de software.

9. Bibliografía

[1] (No date) Manifiesto por el Desarrollo ágil de software. Available at: https://agilemanifesto.org/iso/es/manifesto.html (Accessed: 09 October 2023).

[2] ISO/IEC/IEEE 12207:2017(E) first edition 2017-11: ISO/IEC/IEEE International Standard - Systems and software engineering -- software life cycle processes (2017). IEEE.

[3] Sánchez Méndez. (2013). Modelo y prácticas esenciales de la metodología dac integrando los métodos ágiles, pmbok y cmmi-dev. QUID: Investigación, Ciencia y Tecnología, 21, 13–24.

[4] Fowler, & Scott, K. (2000). UML distilled: a brief guide to the standard object modeling language / Martin Fowler with Kendall Scott; foreword by Grady Booch, Ivar Jacobson, and James Rumbaugh (Second edition.). Addison Wesley Professional.

[5] Rubin. (2013). Essential Scrum: a practical guide to the most popular agile process (1st edition). Addison Wesley.

[6] Humble, J. and Farley, D. (2015) Continuous delivery: Reliable software releases through build, test, and Deployment Automation. Upper Saddle River, NJ u.a: Addison-Wesley.

[7] Granell, A., Medina, M.G.H. and Rosillo, A. (no date) Mantenimiento correctivo, preventivo y predictivo del coche... ¿en qué se diferencian?, RO. Available at: https://www.ro-des.com/blog/mantenimiento-correctivo-preventivo-y-predictivo-del-coche-en-que-se-diferencian/ (Accessed: 11 October 2023).

[8] Ingeniería de software (2023) Wikipedia. Available at: https://es.wikipedia.org/wiki/Ingenier%C3%ADa_de_software (Accessed: 11 October 2023).

[9] Selleri Silva, Soares, F. S. F., Peres, A. L., Azevedo, I. M. de, Vasconcelos, A. P. L. F., Kamei, F. K., & Meira, S. R. de L. (2015). Using CMMI together with agile software development: A systematic review. Information and Software Technology, 58, 20–43. https://doi.org/10.1016/j.infsof.2014.09.012

[10] Henriquez, Calvo-Manzano, J. A., Moreno, A. M., & San Feliu, T. (2022). Agile-CMMI V2.0 alignment: Bringing to light the agile artifacts pointed out by CMMI. Computer Standards and Interfaces, 82, 103610–. https://doi.org/10.1016/j.csi.2021.103610

[11] Cmmi Institute (no date) CMMI Institute - CMMI V2.0 Development Model. Available at: https://cmmiinstitute.com/resource-

files/public/cmmi-v2-0-development-model (Accessed: 21 October 2023).

[12] Cmmi Institute (no date a) CMMI Institute - Guía de adopción y transición de CMMI V2.0 (Version 2.1). Available at: https://cmmiinstitute.com/resource-files/public/cmmi-model-materials/guia-de-adopcion-y-transicion-de-cmmi-v2-0-(versio (Accessed: 22 October 2023).

[13] (No date) Modelo cmmi Y M Etodos Agiles en la gesti on de proyectos ... - uniovi.es. Available at: https://digibuo.uniovi.es/dspace/bitstream/handle/10651/43638/TFMJuanAlonsoBaldonedoRUO.pdf?sequence=3 (Accessed: 27 October 2023).

[14] Richards, M. and Ford, N. (2023) Fundamentals of Software Architecture: An Engineering Approach. Sebastopol, CA: O'Reilly Media, Inc.

[15] Contributor, T. (2016) What is walking skeleton?: Definition from TechTarget, WhatIs.com. Available at: https://www.techtarget.com/whatis/definition/walking-skeleton (Accessed: 08 November 2023).

[16] Miquel, J.P. i and Antonio, R.M.J. (2016) Ingeniería de requisitos. Barcelona: FUOC.

[17] Dev), D.S. (Dan T. (2021) Walking skeleton: Beginners tips, Medium. Available at: https://medium.com/dan-the-dev/walking-skeleton-beginners-tips-deef5baebb5b (Accessed: 08 November 2023).

[18] A successful Git branching model (no date) nvie.com. Available at: https://nvie.com/posts/a-successful-git-branching-model/ (Accessed: 08 November 2023).

[19] Preston-Werner, T. (no date) Versionado Semántico 2.0.0, Semantic Versioning. Available at: https://semver.org/lang/es/ (Accessed: 08 November 2023).

[20] (No date) Redirecting... Available at: https://docs.gitlab.com/ee/topics/gitlab_flow.html (Accessed: 08 November 2023).

[21] Microservices pattern: A pattern language for microservices (no date) microservices.io. Available at: https://microservices.io/patterns/index.html (Accessed: 09 November 2023).

[22] Humble, J. and Farley, D. (2014) Continuous delivery. Upper Saddle River: Addison-Wesley.

[23] Richardson, C. (2019) Microservices patterns: With examples in Java. Shelter Island: Manning Publications.

[24] Awesome observability (no date) University of SRE A Practical Deep Drive. Available at: https://sreuniversity.in/monitoring-and-observability/ (Accessed: 11 November 2023).

[25] Concordion vs selenium comparison of testing frameworks what are the differences between concordion and selenium? (no date) Concordion vs Selenium comparison of testing frameworks. Available at: https://knapsackpro.com/testing_frameworks/difference_between/concordion/vs/selenium (Accessed: 11 November 2023).

[26] Learning, A.O. (2023) Spring security, Medium. Available at: https://medium.com/@pdelvalle9797/spring-security-f47179300d40 (Accessed: 11 November 2023).

[27] Team, K. (2022) Ventajas de Jenkins, KeepCoding Bootcamps. Available at: https://keepcoding.io/blog/ventajas-de-jenkins/ (Accessed: 11 November 2023).

[28] Chacón, J. and Leal, C. (2022) Cuaderno de diseño y prototipaje de una propuesta de mejora de una interfaz.

[29] Gordadze, I. (2020) Spring security jwt tutorial: Toptal®, Toptal Engineering Blog. Available at: https://www.toptal.com/spring/spring-security-tutorial (Accessed: 02 December 2023).

[30] Spring Boot Security + JWT hello world example (no date) JavaInUse. Available at: https://www.javainuse.com/spring/boot-jwt (Accessed: 02 December 2023).

[31] Sevestre, W. by: P. (2023) Using Spring Cloud Gateway with oauth 2.0 patterns, Baeldung. Available at: https://www.baeldung.com/spring-cloud-gateway-oauth2 (Accessed: 02 December 2023).

[32] ¿Cómo crear el login? spring boot 3 + spring security 6 + JWT authentication #backend (2023) YouTube. Available at: https://www.youtube.com/watch?v=nwqQYCM4YT8 (Accessed: 02 December 2023).

[33] Spring Security 6. Entendiendo Los Componentes y el flujo de trabajo. #backend (2023) YouTube. Available at: https://www.youtube.com/watch?v=qiPh0yrDNas (Accessed: 02 December 2023).

[34] Zapater, S. (2023) BDD testing. ¿Cómo funciona el behavior driven development?, Blog de hiberus. Available at: https://www.hiberus.com/crecemos-contigo/bdd-behavior-driven-developement/ (Accessed: 28 December 2023).

[35] Gherkin para Escribir Historias de Usuario (no date) Thiga España. Available at: https://www.media.thiga.co/es/gherkin#:~:text=Gherkin%20es%20un%20lenguaje%20de,de%20una%20historia%20de%20usuario. (Accessed: 28 December 2023).

[36] BDD (behaviour driven development) (no date) Thiga. Available at: https://www.media.thiga.co/es/glosario/bdd-behaviour-driven-development (Accessed: 28 December 2023).

[37] Microservices pattern: Transactional outbox (no date) microservices.io. Available at: https://microservices.io/patterns/data/transactional-outbox.html (Accessed: 28 December 2023).

[38] UOC (2022) 'PRAC1_ISCSD_solucion'.

[39] Eventuate-Tram (no date) Eventuate-tram/eventuate-tram-core: Transactional messaging for microservices, GitHub. Available at: https://github.com/eventuate-tram/eventuate-tram-core (Accessed: 28 December 2023).

[40] Comunicando dos Microservicios USANDO Apache kafka (2022) YouTube. Available at: https://www.youtube.com/watch?v=1dvCwFgHZCk (Accessed: 28 December 2023).

[41] Deagreez et al. (no date) Foto de Cuerpo Completo de sobrepeso Despreocupado Delicioso extático Activo Activo Envivo Escuchar Música hip-hop chic Abuelo Señalando Con el Dedo Hacia Arriba disfrutando del movimiento del ritmo tiene gran vientre Aislado Fondo vívido - stock foto e Imagen De Stock, iStock. Available at: https://www.istockphoto.com/es/search/2/image-film?phrase=papa+noel+navidad (Accessed: 28 December 2023).

10. Anexos

[Anexo 1] 013 Mantenimiento AURIS_COROLLA Toyota Relax 2022

PROGRAMA DE MANTENIMIENTO AURIS / COROLLA SEDÁN

SLLC

Edad (años)	1	2	3	4	5	6	7	8	9	10	11	12
Km	15.000	30.000	45.000	60.000	75.000	90.000	105.000	120.000	135.000	150.000	165.000	180.000
OPERACIONES COMUNES A TODOS LOS MODELOS												
Alfombrillas	V	V	V	V	V	V	V	V	V	V	V	V
Recorrido del pedal de freno, funcionamiento del freno de mano y juego del volante.		V		V		V		V		V		V
Comprobación del estado de la batería.	V	V	V	V	V	V	V	V	V	V	V	V
Funcionamiento de luces, indicadores del cuadro, interruptores de puerta y bocina.	V	V	V	V	V	V	V	V	V	V	V	V
Elementos limpia-lava parabrisas delantero y trasero (incl. llenado de líquido) y luneta térmica.	V	V	V	V	V	V	V	V	V	V	V	V
Filtro de polen del aire acondicionado.	V	R	V	R	V	R	V	R	V	R	V	R
Visualmente la posible corrosión de la carrocería y bajos del vehículo.		V		V		V		V		V		V
Componentes de la dirección (cremallera, rótulas,...).		V		V		V		V		V		V
Guardapolvos de los semiejes de transmisión.		V		V		V		V		V		V
Funcionamiento de la suspensión delantera y trasera, estado de rótulas y amortiguadores.		V		V		V		V		V		V
Tubos de escape y sus soportes.		V		V		V		V		V		V
Tuberías hidráulicas del sistema de frenos y cables del freno de mano.		V		V		V		V		V		V
Desgaste de las pastillas y discos de freno delanteros.	V	V	V	V	V	V	V	V	V	V	V	V
Desgaste de las pastillas y discos de freno traseros, si el vehículo los equipa.	V	V	V	V	V	V	V	V	V	V	V	V
Tapón del depósito de combustible, tuberías, conexiones y válvula de control de vapores.		V		V		V		V		V		V
Presión de los neumáticos, incluyendo la rueda de repuesto (si equipa).	V	V	V	V	V	V	V	V	V	V	V	V
Aceite del motor.	R	R	R	R	R	R	R	R	R	R	R	R
Filtro de aceite del motor.	R	R	R	R	R	R	R	R	R	R	R	R
Líquido de frenos (incluido sistema CSC en vehículos equipados con ello).		R		R		R		R		R		R
Estanqueidad del circuito de refrigeración / calefacción.		V		V		V		V		V		V
Estado y tensión de las correas transmisoras **(excepto Auris Hybrid).**							V	V	V	V	V	V
Líquido refrigerante del motor. Después cada 90K.		V		V		V		V		R		V
Líquido refrigerante del inversor **(Sólo Auris Hybrid, reemplazo a los 240.000 kms).**		V		V		V		V		V		V
Líquido refrigerante del intercooler **(Sólo motor 1.2T).** Después, cada 80K.		V		V		V		V		R		V
Filtro de aire de motor	V	R	V	R	V	R	V	R	V	R	V	R
Inspección Integral / Diagnosis electrónica	I	I	D	D	D	D	D	D	D	D	D	D
Extensión de garantía Toyota Relax (hasta 1 día antes del 10º año o 160.000 Km)			Ext	Ext	Ext	Ext	Ext	Ext	Ext	Ext		
Filtro sistema ventilación batería HV **(Limpieza cada 15.000 sólo hybrid uso intensivo Taxi)**	L	R	L	R	L	R	L	R	L	R	L	R
MODELOS CON MOTOR DE GASOLINA												
Bujías **(sólo Auris 1.4 VVT-i).**		V		R		V		R		V		R
Bujías con electrodo de iridio **(Todos excepto 1.4 VVT-i)**						R						R
Depósito de carbón activo (cánister).			V			V			V			V
Holgura de válvulas con estetoscopio **(sólo Auris 1.4 VVT-i)**						V						V
MODELOS CON MOTOR DIESEL												
Filtro de combustible.			R			R			R			R
Emisión de humos.			V			V			V			V
Holgura de válvulas con estetoscopio **(sólo Auris 1.4 D4D).**			V			V			V			V
MODELOS CON CAMBIO MANUAL												
Líquido de embrague.		V		V		V		V		V		V
Palanca de cambios de 6 velocidades **(Auris 1.33 VVT-i).**		V		V		V		V		V		V
Aceite de la transmisión manual.				V				V				V
MODELOS CON CAMBIO AUTOMÁTICO												
Aceite de la transmisión automática tipo T-IV.			V			V			V			V
Aceite de la transmisión automática tipo CVTF FE **(Modelos con CVT)**				V				V				V
Aceite del transeje híbrido tipo WS **(Auris Hybrid uso intensivo sustituir cada 90.000Km)**			V			V			V			V

69	121	99	189	99	151	99	189	99	216	69	159	€	1.4 VVT-i	ZZE150 (97 CV)	4ZZ-FE	01/03/2007 - 01/11/2008	AURIS / COROLLA SEDÁN
0,5	1,3	1,0	1,5	0,8	1,8	0,8	1,5	1,0	1,7	0,5	1,4						
76	128	106	158	106	230	106	158	106	219	76	200	€	1.6 Valvematic y 1.6 Valvematic MMT	ZRE151 (124 CV)	1ZR-FE	01/03/2007 - 01/06/2009	
0,5	1,1	1,0	1,4	0,8	1,7	0,9	1,4	1,0	1,5	0,6	1,4						
77	135	140	165	107	198	107	165	140	224	77	168	€	1.4 D4D y 1.4 D4D MMT	NDE150 (90 CV)	1ND-TV	01/03/2007 - 01/04/2009	
0,6	1,2	1,5	2,0	0,9	2,0	1,0	2,1	1,5	1,6	0,8	1,8						
105	163	168	193	135	226	135	193	168	274	105	196	€	2.0 D4D	ADE150 (126 CV)	1AD-FTV	01/03/2007 - 01/06/2009	
0,6	1,2	0,9	2,0	0,9	1,5	0,9	2,1	0,9	1,6	0,6	1,8						
100	158	162	188	130	220	130	188	162	268	100	190	€	2.2 D4D	ADE157 (177 CV)	2AD-FHV	01/03/2007 - Fin	
0,6	1,2	0,9	2,0	0,9	1,5	0,9	2,1	0,9	1,6	0,6	1,8						
65	117	95	147	95	218	95	147	95	198	65	188	€	1.33 VVT-i	NRE150 (101 CV)	1NR-FE	01/11/2008 - 30/11/2012	
0,6	1,3	1,1	1,6	0,9	1,9	0,9	1,6	1,1	1,7	0,6	1,6						
105	163	168	193	135	226	135	193	168	274	105	196	€	2.0 D4D	ADE150 (126 CV)	1AD-FTV DPF	01/06/2009 - 30/11/2012	
0,6	1,2	0,9	2,0	0,9	1,5	0,9	2,1	0,9	1,6	0,6	1,8						
76	128	106	158	106	230	106	158	106	219	76	200	€	1.6 Valvematic, 1.6 Valvematic ConfortDrive	ZRE151 (132 CV)	1ZR-FAE	01/06/2009 - 30/11/2012	
0,5	1,2	1,0	1,5	0,8	1,8	0,9	1,5	1,0	1,7	0,6	1,5						
72	130	133	160	102	191	102	160	133	220	72	161	€	1.4 D4D	NDE150 (90 CV)	1ND-TV	01/04/2009 - 01/03/2010	
0,6	1,2	1,5	2,0	0,9	2,0	1,0	2,1	1,5	1,6	0,8	1,8						
76	128	106	158	106	230	106	158	106	219	76	200	€	1.4 D4D y 1.4 D4D MMT	NDE150 (90 CV)	1ND-TV DPF	01/03/2010 - 30/11/2012	
0,6	1,2	1,5	2,0	0,9	2,0	1,0	2,1	1,5	1,6	0,8	1,8						
76	142	106	172	106	243	106	172	106	242	76	213	€	1.8 VVT-i Hybrid	ZWE150	2ZR-FXE	01/03/2010 - 30/11/2012	
0,5	1,1	1,0	1,4	0,8	1,7	0,8	1,4	1,0	1,5	0,5	1,4						

76	128	106	158	106	230	106	158	106	219	76	200	€	1.6 Valvematic, 1.6 Valvematic	ZRE185 (132 CV)	1ZR-FAE	01/12/2012 - 01/06/2015	AURIS serie 180
0,6	1,3	1,0	1,6	0,9	1,9	0,9	1,6	1,0	1,7	0,6	1,6						
72	130	134	160	102	192	102	160	134	220	72	162	€	1.4 D4D y 1.4 D4D MMT	NDE180 (90 CV)	1ND-TV	01/12/2012 - 01/05/2015	
0,5	1,2	0,9	1,8	0,8	1,5	0,9	1,8	0,9	1,6	0,6	1,5						
69	127	131	157	99	190	99	157	131	218	69	160	€	1.4 D4D y 1.4 D4D MMT	NDE180 (90 CV)	1ND-TV	01/05/2015 -	
0,5	1,2	0,9	1,8	0,8	1,5	0,9	1,8	0,9	1,6	0,6	1,5						
100	158	162	188	130	220	130	188	162	268	100	190	€	2.0 D4D	ADE186 (124 CV)	1AD-FTV	01/12/2012 - 01/06/2015	
0,5	1,2	0,8	1,9	0,8	1,5	0,9	1,9	0,9	1,6	0,6	1,6						
76	145	106	175	106	284	106	175	106	233	76	254	€	1.8 VVT-i Hybrid	ZWE186	2ZR-FXE	01/12/2012 - 01/12/2018	
0,6	1,2	1,0	1,5	0,9	1,7	0,9	1,5	1,0	1,6	0,6	1,4						
73	131	103	161	103	233	103	161	103	253	73	203	€	1.2 Turbo	NRE185 (116 CV)	8NR-FTS	01/06/2015 - 01/12/2018	
0,4	1,3	0,9	1,6	0,7	1,8	0,8	1,6	0,9	1,8	0,5	1,6						

76	145	106	175	106	284	106	175	106	233	76	254	€	1.8 Hybrid	ZWE211	2ZR-FXE	01/01/2019 -	COROLLA serie 200
0,5	1,3	1,0	1,6	0,8	1,9	0,8	1,6	1,0	1,6	0,5	1,6						
77	155	107	185	107	294	107	185	107	255	77	264	€	2.0 Hybrid	MZEH12	M20-FXE	01/01/2019 -	
0,5	1,3	1,0	1,6	0,8	1,9	0,8	1,6	1,0	1,6	0,5	1,6						

Tarifas vigentes desde Abril de 2022

Figura 76 Programa de Mantenimiento Toyota Auris / Corolla Sedán

PROGRAMA DE MANTENIMIENTO AURIS 1WW

PLLC

Edad (años) / Km												OPERACIONES COMUNES A TODOS LOS MODELOS
1	2	3	4	5	6	7	8	9	10	11	12	
20.000	40.000	60.000	80.000	100.000	120.000	140.000	160.000	180.000	200.000	220.000	240.000	
				V	V	V	V	V	V	V	V	Estado y tensión de la correas transmisoras
R	R	R	R	R	R	R	R	R	R	R	R	Aceite de motor (ACEA C3). **Sustituir cada 20.000 kms o cuando lo indique el cuadro, lo que primero ocurra.**
R	R	R	R	R	R	R	R	R	R	R	R	Filtro aceite de motor
	V		V		V		V		V		V	Estanqueidad del circuito de refrigeración / calefacción
V	V	V	V	V	V	V	R	V	V	V	R	Líquido refrigerante del motor **(Premium LLC)**
V	V	V	V	V	V	V	V	V	V	V	V	Sistema de escape y sus soportes
V	V	R	V	V	R	V	V	R	V	V	R	Filtro de combustible
		V			V			V			V	Emisión de humos diesel
		R			R			R			R	Filtro de aire de motor
V	V	V	V	V	V	V	V	V	V	V	V	Tapón del depósito de combustible, tubos de combustible y conexiones
I	I	I	I	I	I	I	I	I	I	I	I	Sistema OMMS cambio de aceite
V	V	V	V	V	V	V	V	V	V	V	V	Alfombrillas
	V		V		V		V		V		V	Recorrido del pedal de freno, funcionamiento del freno de mano
	V		V		V		V		V		V	Tuberías y zapatas del freno de estacionamiento
V	V	V	V	V	V	V	V	V	V	V	V	Desgaste de las pastillas y discos de freno delanteros y traseros
V	V	V	V	V	V	V	V	V	V	V	V	Líquido de embrague
	R		R		R		R		R		R	Líquido de frenos
V	V	V	V	V	V	V	V	V	V	V	V	Tuberías hidráulicas del sistema de frenos y cables del freno de mano
V	V	V	V	V	V	V	V	V	V	V	V	Componentes de la dirección (volante, cremallera, rótulas,...)
V	V	V	V	V	V	V	V	V	V	V	V	Guardapolvos de los semiejes de transmisión
V	V	V	V	V	V	V	V	V	V	V	V	Funcionamiento de la suspensión delantera y trasera, estado de rótulas, fuelles y amortiguadores
	V		V		V		V		V		V	Palanca de cambios de 6 velocidades
		V			V			V			V	Aceite de la transmisión manual **(LV API GL-4)**
		V			V			V			V	Aceite del diferencial delantero **(LV API GL-4)**
V	V	V	V	V	V	V	V	V	V	V	V	Estado y presión de los neumáticos
R	R	R	R	R	R	R	R	R	R	R	R	Filtro de polen del aire acondicionado
I	I	D	D	D	D	D	D	D	D	D	D	Inspección Integral / Diagnosis electrónica
		Ext	Ext	Ext	Ext	Ext	Ext	Ext	Ext			Extensión de garantía Toyota Relax (hasta 1 día antes del 10º año o 160.000 Km)
V	V	V	V	V	V	V	V	V	V	V	V	Visualmente la posible corrosión de la carrocería y bajos del vehículo

124	130	220	160	154	226	154	235	220	160	124	271	€	1.6 D	WWE185 (111 CV)	1WW	01/06/2015 - 01/12/2018	AURIS
1,1	1,4	1,6	1,7	1,4	2,0	1,4	1,9	1,7	1,5	1,1	1,6						

Figura 77 Programa de mantenimiento Auris 1WW

[Anexo 2] Documentación de Análisis y Diseño de la Arquitectura.pdf

[Anexo 3] Documentación de Análisis y Diseño de la gestión de usuarios.pdf

[Anexo 4] Documentación de Análisis y Diseño de la gestión de tareas.pdf

[Anexo 5] Documentación de Análisis y Diseño de la gestión de notificaciones.pdf

[Anexo 6] Backlog de Jira

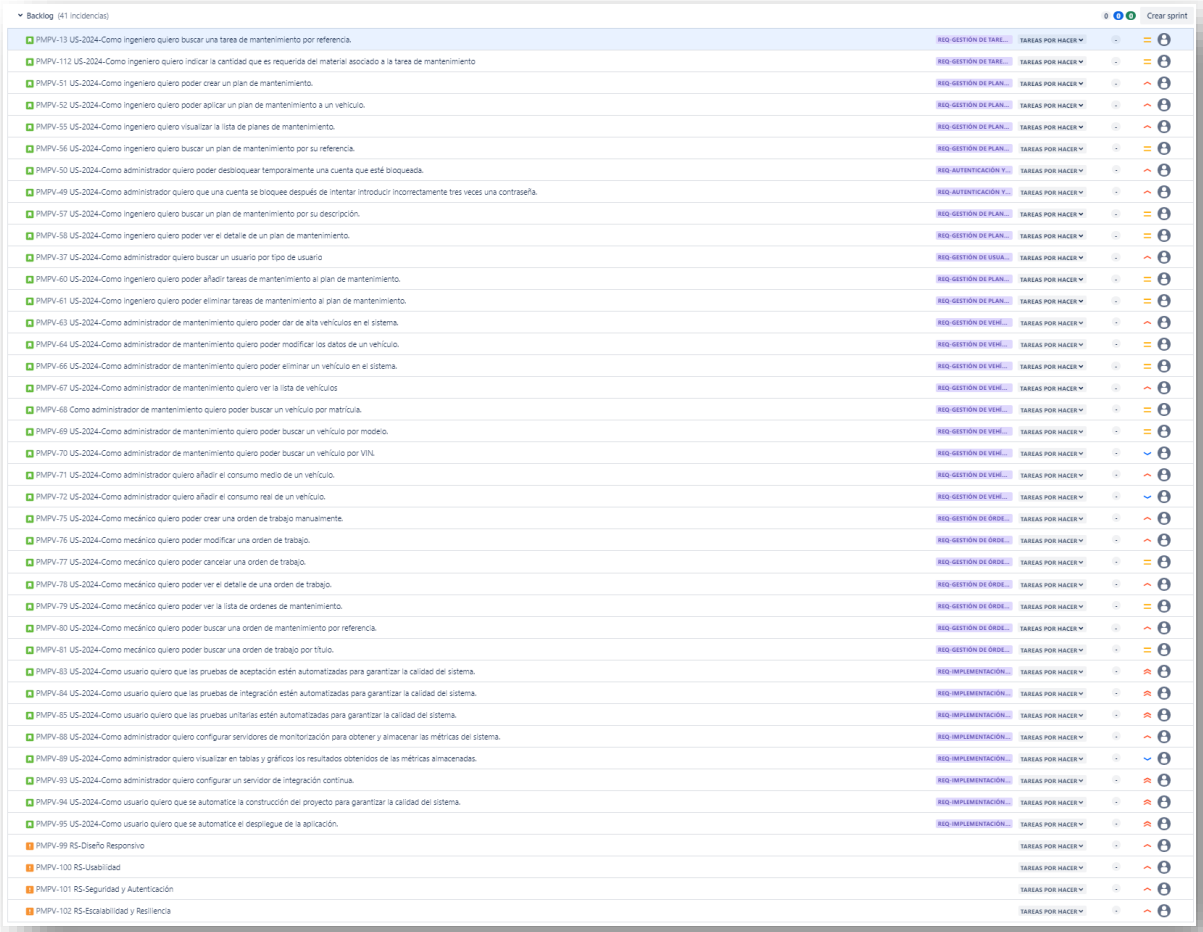

Figura 78 Backlog de Jira