EDUCACIÓN
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLÓGICO NACIONAL DE MÉXICO

Instituto Tecnológico de Acapulco

TECNOLÓGICO NACIONAL DE MÉXICO
INSTITUTO TECNOLÓGICO DE ACAPULCO

DESARROLLO DE UNA PLATAFORMA INFORMÁTICA QUE IMPLEMENTA LA METODOLOGÍA SCRUM PARA GESTIONAR LA PRODUCCIÓN DE SOFTWARE

TITULACIÓN INTEGRAL

TESIS PROFESIONAL

QUE PARA OBTENER EL TÍTULO DE:
MAESTRO EN SISTEMAS COMPUTACIONALES

PRESENTA:
ING. JOSÉ RAÚL LÓPEZ MORALES

DIRECTOR
M.C. JOSÉ FRANCISCO GAZGA PORTILLO

CODIRECTOR
M.T.I. JUAN MIGUEL HERNÁNDEZ BRAVO

TUTOR
M.I.D.S. ALMA DELIA DE JESÚS ISLAO

ACAPULCO GRO. DICIEMBRE 2020

El presente trabajo de tesis fue desarrollado en la *División de Estudios de Posgrado e Investigación del Instituto Tecnológico de* Acapulco, en el programa de Maestría en Sistemas Computacionales perteneciente al Programa Nacional de Posgrado de Calidad (PNPC-CONACyT).

Con domicilio para recibir y oír notificaciones en AV. Instituto Tecnológico de Acapulco s/n, Crucero del Cayaco, Acapulco, Guerrero, México. C.P. 39905.

Becario:	José Raúl López Morales.
CVU:	928537.
Núm. de apoyo:	711644.
Grado:	Maestría.

Descargo de responsabilidad Institucional

Quien suscribe declara que el presente documento titulado “Desarrollo de una plataforma informática que implementa la metodología Scrum para gestionar la producción de software” es un trabajo propio y original, el cual no ha sido utilizado anteriormente en institución alguna para propósitos de evaluación, publicación y/o obtención de algún grado académico.

Además, se adelanta que se han recogido todas las fuentes de información utilizadas, las cuales han sido citadas en la sección de referencias bibliográfica de este trabajo.

Acapulco, Gro; a 15 de diciembre de 2020.

Ing. José Raúl López Morales

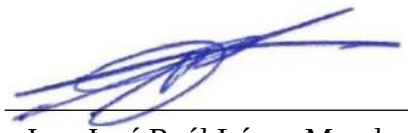
Carta de cesión de derechos de autor

El que suscribe: José Raúl López Morales, autor del trabajo escrito de evaluación profesional en la opción de Tesis de Maestría con título “Desarrollo de una plataforma informática que implementa la metodología Scrum para gestionar la producción de software”, por medio de la presente con fundamento en lo dispuesto en los artículos 5, 18, 24, 25, 27, 30, 32 y 148 de la Ley Federal de Derechos de Autor, así como los numerales 2.15.5 de los lineamientos para la Operación de los Estudios de Posgrado; manifiesto mi autoría y originalidad de la obra mencionada que se presentó en la División de Estudios de Posgrado e Investigación, para ser evaluada con el fin de obtener el Título Profesional de Maestro en Sistemas Computacionales.

Así mismo expreso mi conformidad de ceder los derechos de reproducción, difusión y circulación de esta obra, en forma NO EXCLUSIVA, al Tecnológico Nacional de México campus Acapulco; se podrá realizar a nivel nacional e internacional, de manera parcial o total a través de cualquier medio de información que sea susceptible para ello, en una o varias ocasiones, así como en cualquier soporte documental, todo ello siempre y cuando sus fines sean académicos, humanísticos, tecnológicos, históricos, artísticos, sociales, científicos u otra manifestación de la cultura.

Entendiendo que dicha cesión no genera obligación alguna para el Tecnológico Nacional de México campus Acapulco y que podrá o no ejercer los derechos cedidos. Por lo que el autor da su consentimiento para la publicación de su trabajo escrito de evaluación profesional.

Se firma presente en la ciudad de Acapulco de Juárez, estado de Guerrero a los 15 días del mes de diciembre de 2020.



Ing. José Raúl López Morales

EDUCACIÓN
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLÓGICO NACIONAL DE MÉXICO

Instituto Tecnológico de Acapulco
División de Estudios de Posgrado e Investigación

"2020, Año de Leona Vicario, Benemérita Madre de la Patria"

Acapulco, Gro; a 07 de diciembre de 2020.

AUTORIZACIÓN DE IMPRESIÓN DE TESIS

Los abajo firmantes, miembros de la comisión revisora de tesis designada por la División de Estudios de Posgrado e Investigación del Tecnológico Nacional de México campus Acapulco para la evaluación de la tesis del alumno **José Raúl López Morales**, manifiestan que después de haber revisado su tesis: **"Desarrollo de una plataforma informática que implementa la metodología Scrum para gestionar la producción de software"** desarrollada bajo la dirección del DIRECTOR, y el CO-DIRECTOR, el trabajo se **ACEPTA** para proceder a su impresión.

A T E N T A M E N T E

M.C. José Francisco Gazga Portillo
Cédula Profesional: 5243159

M.T.I. Juan Miguel Hernández Bravo
Cédula Profesional: 6237953

M.I.D.S. Alma Delia de Jesús Islao
Cédula Profesional: 8604126

SECRETARÍA DE EDUCACIÓN PÚBLICA
INSTITUTO TECNOLÓGICO DE ACAPULCO
DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

Enterado

Dr. Eduardo de la Cruz Gámez
Coordinador de la Maestría en Sistemas Computacionales

EDUCACIÓN
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLÓGICO NACIONAL DE MÉXICO

Instituto Tecnológico de Acapulco
División de Estudios de Posgrado e Investigación

"2020, Año de Leona Vicario, Benemérita Madre de la Patria"

Acapulco Gro., 8/Diciembre/2020

NO. OFICIO: DEPI-206/2020

ASUNTO:
AUTORIZACIÓN DE
IMPRESIÓN DE TESIS PROFESIONAL

C. JOSÉ RAÚL LÓPEZ MORALES

De acuerdo al reglamento de los Institutos Tecnológicos, dependiente de la Secretaría de Educación Pública y habiendo cumplido con todos los requisitos normativos respecto a su trabajo para titulación, Opción Titulación Tesis Profesional, con el proyecto titulado: DESARROLLO DE UNA PLATAFORMA INFORMÁTICA QUE IMPLEMENTA LA METODOLOGÍA SCRUM PARA GESTIONAR LA PRODUCCIÓN DE SOFTWARE. Se **CONCEDE** la **AUTORIZACIÓN** para que proceda a la impresión del mismo.

Sin otro particular por el momento, me es grato quedar de usted.

A T E N T A M E N T E "
Educación Tecnológica con Compromiso Social"

SECRETARÍA DE EDUCACIÓN PÚBLICA
INSTITUTO TECNOLÓGICO DE ACAPULCO
DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

EDUARDO DE LA CRUZ GÁMEZ
JEFE DE LA DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

C.c.p. Expediente

EDG/stv

Agradecimientos

Agradezco primero y ante todo a Dios, todo se lo debo a Él, por derramar sus Bendiciones sobre mí, por acompañarme en cada paso que doy y haber fortalecido mi corazón e iluminado mi mente. Gracias Padre.

A mi familia, que ha estado ahí, siempre a mi lado en los momentos más oscuros y difíciles creyendo en mí, impulsándome para alcanzar mis metas.

Al Consejo Nacional de Ciencia y Tecnología por haberme apoyado económicamente para la realización de este proyecto, dándome la oportunidad de seguir conociendo y aprendiendo.

A mi director de tesis, el M.C. José Francisco Gazga Portillo por su tiempo, por su paciencia, por los valiosos aportes, críticas, comentarios y sugerencias que realizó durante este proyecto.

Al departamento de Posgrado e Investigación del Instituto Tecnológico de Acapulco, así como a toda la plantilla docente, de la cual me llevo conocimiento y experiencias enriquecedoras que sin lugar a dudas marcaron mi vida profesional.

Dedicatoria

A mis compañeros y amigos José Alberto, Alejandro, Rogelio (team hugitos) y Ángel Rayo, quienes me apoyaron su momento para salir de un apuro en la maestría, así como hacer más amena mi estancia en el departamento de Posgrado e Investigación del Instituto Tecnológico de Acapulco.

A mi amiga Crisol Angelina que en los últimos meses de la maestría me ha orientado en mis procesos, tanto de la coordinación o propios, así como estar ahí para recordarme sobre mis pendientes y de vez en cuando alimentarme.

Al Ing. Hugo Rojas por sus enseñanzas en cuestiones de desarrollo de software, haciendo posible el desarrollo de este proyecto. Así como proporcionarme herramientas y conocimiento para enfrentarme en el campo laboral.

A Yazmin Hernández Suastegui por su apoyo antes y durante mi estancia en la maestría, y recordándome de lo que soy capaz.

Resumen

En el presente trabajo se describe el desarrollo de una plataforma informática que permite gestionar y monitorizar proyectos de desarrollo de software basados en la metodología Scrum.

La plataforma propuesta está diseñada con base en el patrón arquitectónico MVC, mediante el cual está estructurado el marco de desarrollo ASP.NET Core MVC, este patrón consta de tres componentes: *Modelo*: encapsula o representa los datos de la aplicación el cual es consultado por el **controlador** y a su vez necesitado en la **vista** para ser representado en ella, *Vista*: se encarga de la interacción con el usuario y la representación de un **modelo** mediante una **interfaz gráfica de usuario**, *Controlador*: es el intermediario entre el **modelo** y la **vista** ante las peticiones generadas por el cliente mediante la **vista**.

Además, la plataforma implementa los patrones repositorio y unidad de trabajo, permiten crear una capa de abstracción entre la capa de acceso a datos y la capa de lógica de negocios en la plataforma. Con la implementación de estos patrones se permite aislar la aplicación de los cambios en la persistencia de datos.

La implementación de la plataforma propuesta, permitirá fortalecer el desarrollo de software en el departamento de desarrollo de la empresa: Proyectos, Instalaciones y Construcciones Civiles y Eléctrica (PICCE), al proveer una forma de monitorizar y gestionar los proyectos con base a la metodología Scrum, proporcionando así, una flexibilidad en los cambios de requerimientos, además de entregas funcionales en cortos periodos.

Abstract

This Document describes the development of a computer application that allow managing and monitoring software development projects based on the Scrum methodology.

The proposed platform is designed based on the MVC architectural pattern, through which the ASP.NET Core MVC development framework is structured, this pattern consist of three components: Modelo: it encapsulates or represents the application data which is consulted by the **controller** and in turn needed in the **view** to be represented in it, View: it is responsible for the interaction with the user and the representation of a **model** through a graphical user interface, Controller: it is the intermediary between the **mode** and the **view** before the request generated by the client through the **view**.

In addition, the platform implements the repository and unit of work patterns to create an abstraction layer between the data access layer and the business logic layer in the platform. With the implementation of these patterns, it helps to isolate the application of changes in data persistence.

The implementation of the proposed platform will strengthen the software development in the company's development: Proyectos, Instalaciones y Construcciones Civiles y Eléctrica (PICCE), by providing a way to monitor and manage projects based on the Scrum methodology, providing flexibility in changing requirements and functional deliveries in short periods.

Índice de contenido

Índice de figuras

Índice de tablas

Capítulo 1 Introducción

1.1 Estado del arte

El desarrollo de software no ha sido una tarea simple, por tal motivo existen propuestas metodológicas que inciden en distintas partes del proceso de desarrollo. Por otra parte, existen las metodologías tradicionales las cuales se centran especialmente en el control del proceso, estableciendo rigurosamente actividades, los artefactos que se deben producir y las herramientas que usarán (Letelier & Panadés, 2012). Las metodologías tradicionales como modelo en cascada, prototipo, incremental, espiral, desarrollo rápido de aplicaciones (en inglés Rapid Application Development o RAD), normalmente, especifican por completo los requerimientos y después continúan con las demás etapas involucradas, esto quiere decir que no están orientadas al desarrollo ágil de software. Los requerimientos cambian, o se encuentran problemas en ellos, lo que produce una reelaboración del diseño o la implementación del software. Como consecuencia, una metodología tradicional en cascada o uno basado en especificación, prolonga su entrega mucho después de lo establecido en un principio.

Las metodologías tradicionales han demostrado ser efectivas en gran número de proyectos, pero también han presentado problemas en otros (Letelier & Panadés, 2012). Las metodologías tradicionales exigen una documentación rígida que en muchas ocasiones los clientes no requieren y prefieren apostar por implementación temprana, no son tolerantes a los cambios o contratiempos en el desarrollo, una manera para poder solucionar este problema en ellas es incluir en los procesos de desarrollo más actividades, más artefactos y más restricciones.

A mediados de los 90’s comenzó a forjarse la definición de desarrollo ágil como una reacción en contra de las metodologías tradicionales, que son consideradas pesadas y rígidas por su carácter normativo y una fuerte dependencia de planeaciones detalladas previas al desarrollo. El uso de las

metodologías de desarrollo ágil o metodologías ágiles son más recientes en la ingeniería de software, las cuales están acaparando gran interés y controversia (Rodríguez, 2008).

Los procesos de desarrollo del software rápido se diseñan para producir rápidamente un software útil. El software no se desarrolla como una sola unidad, sino como una serie de incrementos, y cada uno de ellos incluye una nueva funcionalidad del sistema. Aun cuando existen muchos enfoques para el desarrollo de software ágil, todos ellos comparten algunas características fundamentales:

1. Los procesos de especificación, diseño e implementación están enlazados. No existe una especificación detallada del sistema, y la documentación del diseño se minimiza o es generada automáticamente por el entorno de programación que se usa para implementar el sistema. El documento de requerimientos del usuario define sólo las características más importantes del sistema.
2. El sistema se desarrolla en diferentes versiones. Los usuarios finales y otros colaboradores del sistema intervienen en la especificación y evaluación de cada versión. Ellos podrían proponer cambios al software y nuevos requerimientos que se implementen en una versión posterior del sistema.
3. Las interfaces de usuario del sistema se desarrollan usando con frecuencia un sistema de elaboración interactivo. Esto permite que el diseño de la interfaz se cree rápidamente en cuanto se dibujan y colocan iconos en la interfaz. En tal situación, el sistema puede generar una interfaz basada en la Web para un navegador o una interfaz para una plataforma específica, como Microsoft Windows (Sommerville, 2016).

Existen varias formas de metodologías ágiles como el método de desarrollo de sistemas dinámicos (en inglés Dynamic Systems Development Method o DSDM), Scrum, la programación extrema (en inglés eXtreme Programming o XP) pero todas comparten características similares. La piedra angular de cada rama es la idea de la satisfacción del cliente (Blankenship, Bussa, & Millett, 2011).

Como se mencionó anteriormente, existen diversas metodologías y cada una se enfoca en ciertos aspectos del desarrollo de software, por ejemplo, la programación extrema se centra en la propia programación del producto en cambio la metodología Scrum se enfoca en la administración del proyecto. Para el presente proyecto, es muy importante la metodología Scrum ya que la herramienta que se pretende desarrollar gestionará proyectos que se implementen en dicha metodología.

Existen diversas herramientas para la gestión y monitorización de proyectos de desarrollo de software en internet, las cuales en su mayoría tienen un costo elevando, aunque existen también algunas sin costo alguno, pero tienen ciertas restricciones al momento de su uso. **Sprintometer** es una aplicación para la gestión de proyectos Scrum y XP, ofrece una gran variedad de funciones, el único inconveniente es que no permite gestionar un proyecto de manera colaborativa, por ejemplo, si un usuario está trabajando en la gestión de un proyecto, otros usuarios no podrán visualizar y contribuir sobre este proyecto a menos que utilicen el mismo equipo.

A continuación, se describe **Kunagi**, una herramienta en línea para la gestión de proyectos basados en Scrum, al igual que **Sprintometer** sin costo y de la misma manera sólo se puede administrar un solo proyecto, no es intuitivo en su interfaz, sólo permite usar nombres de equipo ya establecidos, no hay un **seguimiento del sprint** (bloques o iteraciones) el cual es la parte más importante de la metodología Scrum y por último no proporciona gráficos de los avances del proyecto.

Por otro lado, la herramienta Scrum en línea **icescrum** permite una gestión completa de proyectos Scrum, así como equipos, usuarios y proyectos ilimitados, sin embargo, para usar la herramienta es necesario registrarse para un periodo de prueba o pagar una suscripción al sitio.

Así como **icescrum** existen otras herramientas Scrum como **jira**, **versionOne**, **targetProcess**, **Active collab** y **pivotal tracker**, sin embargo, son aplicaciones de escritorio en su mayoría requieren configuración experta y se necesita pagar una suscripción a un precio elevado. Por lo anterior, el propósito del presente trabajo de tesis es de proveer una alternativa que permita a PICCE gestionar y monitorizar proyectos de desarrollo de software basados en la metodología Scrum.

La metodología Scrum se fundamenta en el proceso iterativo y proceso incremental para mejorar la previsibilidad, reduce el error y controla el riesgo que se produce durante el tiempo de desarrollo de software (Khalil & Kotaiah, 2017).

La implementación de la metodología Scrum ha mostrado resultados positivos en el desarrollo ágil de software, algunos de los antecedentes más recientes se encuentran en el artículo con el nombre Applying Scrum to the Army - A Case Study publicado en 2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C) por Luigi Benedicenti, Franco Cotugno, Paolo Ciancarini, Angelo Messina, Witold Pedrycz, Albert Sillitti y Giancarlo Succi sobre el uso de la metodología Scrum para el desarrollo ágil de software en el Departamento de Defensa de los Estados Unidos, donde se obtuvieron resultados positivos a pesar que el desarrollo de software en dicho departamento es una tarea compleja porque las soluciones tienen que ser de muy alta calidad y tienen que satisfacer necesidades muy específicas y complejas. (Benedicenti, y otros, 2016). En otro ámbito de estudio, en la tesis de maestría en ingeniería de software de Diego Alberto Godoy se aborda el diseño de una herramienta de simulación de procesos de desarrollo de software llevados a cabo con la metodología Scrum utilizando dinámica de sistemas (Godoy, 2014).

1.2 Planteamiento del problema

El departamento de desarrollo de software de la empresa Proyectos, Instalaciones y Construcciones Civiles y Eléctricas (PICCE) en los últimos años ha estado usando los modelos de desarrollo de software: prototipo e incremental, pero los han dejado de usar debido a ser robustos, es decir, implica que tienen que realizar una serie de fases e ir documentando cada una de ellas, pero no es permitido hacer cambios en los requerimientos que estaban establecidos al inicio de cada proyecto. Los nuevos requerimientos que se definían se tienen que hacer al final del producto entregable y esto causa que se prolongue la fecha de entrega del producto de manera considerable causando en ocasiones gastos para la empresa.

Actualmente, el departamento de desarrollo de software de la empresa: Proyectos, Instalaciones y Construcciones Civiles y Eléctricas (PICCE) ha estado implementando la metodología ágil Scrum en la gestión del desarrollo de proyectos de software debido a que, se trata de una metodología flexible en actualizaciones de requerimientos y las entregas del producto son más rápidas. Sin embargo, aun implementando dicha metodología, se llegan a presentar problemas de entregas de los **sprints**, originando el problema objeto de estudio y solución en el presente trabajo de investigación: la empresa PICCE no cuenta con una herramienta que realice la monitorización día a día del progreso de cada tarea que está involucrada en el desarrollo de sus productos de software, así como las horas diarias que invierte un desarrollador a dichas tareas. Por tal motivo, no es posible percibir, identificar los inconvenientes que se están presentando en el desarrollo de cada producto y tampoco prevenir retrasos en la entrega de los **sprints**, lo que impide estimar el tiempo requerido para alcanzar el objetivo de cada **sprint**.

1.3 Objetivos

1.3.1 Objetivo General

Desarrollar una plataforma web para la gestión y monitorización de proyectos de desarrollo de software basados en la metodología Scrum para la empresa PICCE.

1.3.2 Objetivos específicos

- Recopilar los requerimientos sobre las necesidades de desarrollo de software de la empresa PICCE, así como también las características de las actividades involucradas en el desarrollo de software basado en la metodología Scrum.
- Seleccionar las herramientas de desarrollo y programación web para la construcción de la plataforma propuesta.
- Definir casos de estudio que permitan verificar la funcionalidad de la plataforma web propuesta, con el fin de validar y evaluar la cobertura de los requerimientos de origen.

1.4 Hipótesis

El desarrollo de una herramienta Scrum permitirá monitorizar la gestión del desarrollo de software, además de llevar el control del progreso de cada sprint permitiendo visualizar gráficamente las estimaciones para alcanzar los objetivos o tareas en cada sprint y a su vez obtener entregas a tiempo.

1.5 Justificación

Este trabajo de tesis propone desarrollar una plataforma para gestionar y monitorizar proyectos de desarrollo de software basados en la metodología Scrum, que el departamento de desarrollo de software de la empresa PICCE puede utilizar en un futuro para la gestión y monitorización de sus productos de software.

Por lo tanto, la implementación de esta plataforma permitirá disminuir costos ya que las plataformas que existen en el mercado requiere de pagar por su uso, en un determinado tiempo, aunque no se haga uso de estas durante el periodo contratado. Con esta propuesta de tesis, la plataforma podrá ser usada en cualquier momento que sea necesaria, además, permitirá gestionar un número ilimitado de proyectos y usuarios involucrados en ellos.

Con este proyecto de tesis, los usuarios no tendrán la necesidad de realizar una instalación y configuración experta para su uso, además no será requerido de un equipo con características de alto rendimiento.

Al tener una plataforma de este tipo, permitirá el control adecuado de las tareas asignadas a los integrantes del equipo de desarrollo y del progreso de cada una de ellas a través de gráficos facilitando la toma de decisiones en caso de existir inconvenientes.

Con esta plataforma se podrá administrar correctamente el recurso humano, es decir, que cada una de las personas involucradas en un proyecto tendrá un rol especifico y tareas específicas con la finalidad de tener mayor productividad y mejor calidad en los productos de software. Así como también la administración de los tiempos en cada una de las tareas para alcanzar el objetivo en cada entrega.

En contraste, las plataformas que existen hoy en día no realizan el seguimiento del **Sprint** detalladamente, es decir, no registran diariamente las horas de trabajo de cada integrante del equipo de desarrollo que invirtió en sus tareas asignadas. Por lo que, de no hacer este seguimiento del **Sprint**, podría causar retrasos importantes en cada entrega del producto.

1.6 Organización de la tesis

El trabajo está estructurado en 6 capítulos:

Capítulo 1: Introducción

En este capítulo se describen los aspectos fundamentales del trabajo de investigación, incluyendo la definición de la problemática, los objetivos, la hipótesis y la justificación de la propuesta.

Capítulos 2: Marco teórico

Constituye los aspectos teóricos de referencia de la investigación. Contiene las aportaciones teóricas y conceptuales relacionadas con el desarrollo del tema tratado en este trabajo de tesis.

Capítulos 3: Análisis y diseño

Con la información disponible, se analiza el proceso o pasos que se siguen en la implementación de la metodología **Scrum**, se identifican los procesos en los que interviene la herramienta a desarrollar y se realiza el diseño de los componentes de la misma.

Capítulo 4: Implementación

Se realiza la codificación de los componentes de software de la herramienta soportada por el diseño presentado en el capítulo 3.

Capítulo 5: Resultados

A partir de diversos casos de estudio, se presentan los resultados de la implementación para ser analizados e interpretados.

Capítulo 6: Conclusiones

Se dan conclusiones finales, se evalúa la hipótesis propuesta inicialmente, se presentan las aportaciones del trabajo, así como trabajos a futuro.

Capítulo 2 Marco teórico

A continuación se describe el marco teórico que da soporte al presente trabajo de investigación, el cual se encuentra dividido en cuatro secciones, en la primera sección se describe las metodologías de desarrollo ágil de software, además se menciona la metodología empleada para el desarrollo de la herramienta Scrum, para la segunda sección se presenta las diferentes arquitecturas de software, seleccionando una de ella para la construcción de la aplicación, en la tercera sección se describen los diagramas de lenguaje unificado de modelado (en inglés Unified Modeling Language o UML) más esenciales para el diseño de la aplicación, posteriormente en la cuarta sección se presentan los conceptos que son necesarios para el desarrollo de software, así como los lenguajes de programación y frameworks que están presentes en el ***front-end*** y ***back-end***, además de tecnologías para la manipulación y almacenamiento de datos.

El primer paso para comenzar el desarrollo de software, consiste en contar con una metodología de desarrollo, que permita seguir una secuencia de pasos para la construcción de un determinado software.

2.1 Metodologías ágiles

Los métodos ágiles son métodos de desarrollo incremental donde los incrementos son mínimos y, por lo general, se crean las nuevas liberaciones del sistema, y cada dos o tres semanas se ponen a disposición de los clientes. Involucran a los clientes en el proceso de desarrollo para conseguir una rápida retroalimentación sobre los requerimientos cambiantes. Minimizan la cantidad de documentación con el uso de comunicaciones informales, en vez de reuniones formales con documentos escritos (Sommerville, 2016).

El punto de partida se establece en las ideas emanadas del **Manifiesto Ágil** tras la reunión de Utah en el 2001, un documento que resume la filosofía ágil estableciendo cuatro valores y doce principios (Blankenship, Bussa, & Millett, 2011). Ver la figura 2.1 y 2.2.

Figura 2.1. El manifiesto ágil (Blankenship, Bussa, & Millett, 2011).

Según el manifiesto de la figura 2.1 se valora:

- **Individuos e interacciones por encima de procesos y las herramientas**: para garantizar una mayor productividad, las metodologías ágiles valoran el recurso humano como el principal factor de éxito. Reconocen que contar con recurso calificado con capacidades técnicas adecuadas, facilidades para adaptarse al entorno, trabajar en equipo e interactuar convenientemente con el usuario, da mayor garantía de éxito que contar con herramientas y procesos rigurosos (Herrera, Valencia, & Luz, 2007).
- ***Software* funcionado por encima de una documentación extensiva**. Este valor acentúa la importancia del producto por encima de la documentación, para el desarrollador, el objetivo es lograr un producto que funcione y que cumpla con las necesidades del cliente y la documentación es un artefacto para cumplir sus objetivos. No se trata de no documentar

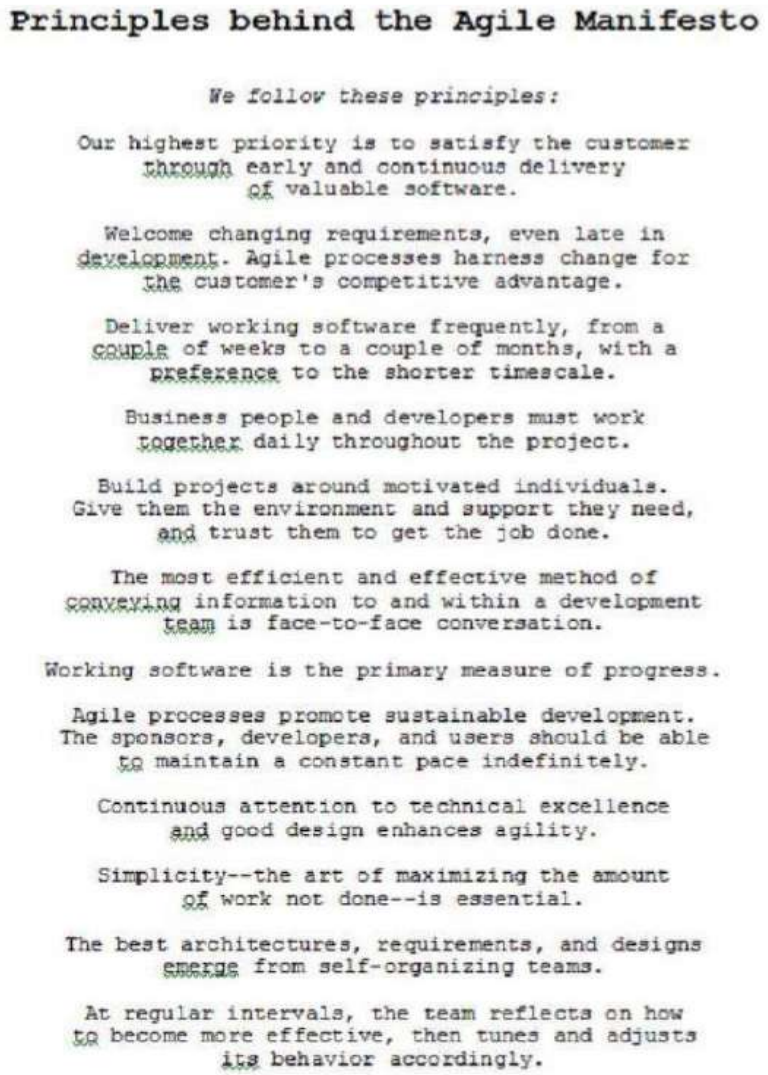
sino documentar lo esencial, esto es para tomar inmediatamente decisiones importantes. Ya que el código es la parte más importante que se obtiene del desarrollo de software, este sigue ciertos estándares de programación para obtener un código legible y documentado.

- **Colaboración con el cliente por encima de la negociación del contrato**. Este valor propone que exista una comunicación continua entre el cliente y el equipo de desarrollo con el objetivo de que el cliente cree un vínculo con el equipo de desarrollo. No pretende distinguir entre los roles cliente y equipo de desarrollo sino ser un solo equipo que tengan un objetivo en común.
- **Respuesta ante el cambio por encima de seguir un plan**. La planificación del trabajo a realizar es muy favorable y las metodologías ágiles consideran una planeación de las actividades específicas a corto plazo. En el desarrollo de software es vital adaptarse a los cambios.

Para cumplir estos valores se siguen doce principios que establecen algunas diferencias entre un desarrollo ágil y uno convencional como se muestra en la figura 2.2. Los principios descritos en la figura 2.2, enuncia lo siguiente:

1. Satisfacer al cliente por medio de la entrega temprana y continua de *software* funcional.
2. Son bienvenidos los requisitos cambiantes, incluso si llegan tarde al desarrollo.
3. Entregar con frecuencia *software* funcional, prefiriendo periodos de semanas y no de meses.
4. Trabajo conjunto y cotidiano entre clientes y desarrolladores.
5. Construcción de proyectos en un ambiente con individuos motivados, dándoles la confianza y el respaldo que necesitan.
6. Comunicación cara a cara como forma eficiente y efectiva de comunicación.
7. El *software* funcional como principal medida de avance.

8. Procesos ágiles como medio para promover el desarrollo sostenido.
9. Atención continua a la excelencia técnica y al buen diseño.
10. Simplicidad como el arte de maximizar la cantidad de trabajo que no se hace.
11. Equipos de desarrollo auto-organizados.
12. Adaptación regular a circunstancias cambiantes (Cervantes Maceda, Velasco-Elizondo, & Castro Careaga, 2016).



Principles behind the Agile Manifesto

We follow these principles:

Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

Business people and developers must work together daily throughout the project.

Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

Working software is the primary measure of progress.

Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

Continuous attention to technical excellence and good design enhances agility.

Simplicity--the art of maximizing the amount of work not done--is essential.

The best architectures, requirements, and designs emerge from self-organizing teams.

At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Figura 2.2. Los doce principios detrás del manifiesto ágil (Blankenship, Bussa, & Millett, 2011).

Aunque los creadores e impulsores de las metodologías ágiles más populares han suscrito el manifiesto ágil y coinciden con los principios mencionados anteriormente, cada metodología tiene

características propias y recalcan en algunos aspectos siendo más específicos. A continuación, se resumen dichas metodologías ágiles, dejando el análisis más detallado de Scrum para la siguiente sección (Letelier & Panadés, 2012).

2.1.1 Método de Cristal (Crystal Method)

Se trata de un conjunto de metodologías para el desarrollo de software caracterizadas por estar centradas en las personas que componen el equipo (de ellas depende el éxito del proyecto) y la reducción al máximo del número de artefactos producidos. Han sido desarrolladas por Alistair Cockburn. El desarrollo de software se considera un juego cooperativo de invención y comunicación, limitado por los recursos a utilizar. El equipo de desarrollo es un factor clave, por lo que se deben invertir esfuerzos en mejorar sus habilidades y destrezas, así como tener políticas de trabajo en equipo definidas. Estas políticas dependerán del tamaño del equipo, estableciéndose una clasificación por colores, por ejemplo, Crystal Clear (3 a 8 miembros) y Crystal Orange (25 a 50 miembros) (Cockburn, 2001).

2.1.2 Método de desarrollo de sistemas dinámicos (Dynamic Systems Development Method, DSDM)

Define el marco para desarrollar un proceso de producción de software. Nace en 1994 con el objetivo de crear una metodología RAD o Desarrollo Rápido de Aplicaciones (en inglés Rapid Application Development) unificada. Sus principales características son: se trata de un proceso iterativo e incremental y el equipo de desarrollo y el usuario trabajan juntos. Propone cinco fases: Estudio Viabilidad, Estudio del Negocio, Modelado Funcional, Diseño y Construcción, y finalmente Implementación. Las tres últimas son iterativas, además de existir retroalimentación a todas las fases (Stapleton, 1997).

2.1.3 Desarrollo adaptativo de software (Adaptive Software Development, ASD)

Su impulsor es Jim Highsmith. Sus principales características son: Iterativo, Orientado a los componentes software más que a las tareas y Tolerante a los cambios. El ciclo de vida que propone tiene tres fases esenciales: Especulación, Colaboración y Aprendizaje. En la primera de ellas se inicia el proyecto y se planifican las características del software; en la segunda desarrollan las características y finalmente en la tercera se revisa su calidad, y se entrega al cliente. La revisión de los componentes sirve para aprender de los errores y volver a iniciar el ciclo de desarrollo (James A. & Orr, 2000).

2.1.4 Desarrollo basado en funcionalidades (Feature-Driven Development, FDD)

Define un proceso iterativo que consta de cinco pasos los cuales son Desarrollar un modelo global, Construir una lista de funcionalidades, Planificar por funcionalidad, Diseñar por funcionalidad y Construir por funcionalidad. Las iteraciones son cortas (hasta dos semanas). Se centra en las fases de diseño e implementación del sistema partiendo de una lista de características que deben reunir el software. Sus impulsores son Jeff De Luca y Peter Coad (Coad, Lefebvre, & De Luca, 1999).

2.1.5 Desarrollo de software lean (Lean Development, LD)

Definida por Bob Charette’s a partir de su experiencia en proyectos con la industria japonesa del automóvil en los años 80 y utilizada en numerosos proyectos de telecomunicaciones en Europa. En LD, los cambios se consideran riesgos, pero si se manejan adecuadamente se pueden convertir en oportunidades que mejoren la productividad del cliente. Su principal característica es introducir un mecanismo para implementar dichos cambios (Poppendieck & Poppendieck, 2003).

2.1.6 Programación extrema (Extreme Programming, XP)

Es una metodología ágil centrada en potenciar las relaciones interpersonales como clave para el éxito en desarrollo de software, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores, y propiciando un buen clima de trabajo. XP se basa en retroalimentación continua entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes, simplicidad en las soluciones implementadas y flexible para enfrentar los cambios. XP se define como especialmente adecuada para proyectos con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico (Beck, 2000).

En la presente propuesta de tesis la metodología ágil a implementar es la metodología Scrum, debido a que la aplicación que se pretende desarrollar gestionará y monitorizará precisamente proyectos basados en la metodología Scrum. Además, está metodología es indicada para proyectos en entornos complejos, la aplicación que se va a desarrollar tiene cierto grado de dificultad y también permite requisitos cambiantes o poco definidos, es decir, es flexible ante cambios y además que las entregas del producto a desarrollar son pequeños incrementos funcionales y estos incrementos se ven reflejados en cortos periodos. A continuación, se describirá la metodología Scrum que da soporte a lo mencionado.

2.1.7 Scrum

Como todo método ágil, Scrum permite de forma iterativa e incremental el desarrollo de *software*. Atendiendo al **Manifiesto ágil**, en Scrum un equipo multifuncional y auto-organizado crea de manera gradual un producto en varias iteraciones cortas. Cada iteración permite inspeccionar el rendimiento del equipo, así como el producto resultante, para luego, si es necesario, llevar a cabo oportunamente las adaptaciones requeridas. Scrum define un conjunto de pequeños roles y un proceso que se describen en las siguientes secciones.

Los roles

Scrum define tres roles principales:

Propietario del producto (product owner): responsable de maximizar tanto el valor del producto que se está construyendo como el trabajo del equipo de desarrollo. Entre sus principales funciones están definir, (re)priorizar y comunicar los requerimientos del producto, así como revisar y aprobar el trabajo y los resultados correspondientes en cada iteración. El propietario debe interactuar activa y regularmente con el equipo.

Equipo de desarrollo (team): grupo multifuncional y auto-organizado de personas encargadas de la construcción del producto. Entre sus principales responsabilidades está decidir el número de requerimientos a desarrollar en una iteración, así como la estrategia para llevarlos a cabo. El equipo se compone por lo habitual de entre 5 y 9 personas y no existen roles más específicos para los integrantes (por ejemplo, el de programador o de diseñador).

Scrum master: responsable de asegurar que el marco de trabajo de Scrum sea entendido y adoptado por todos los involucrados. Puede verse de esta forma como un facilitador para el propietario del producto y el equipo de desarrollo. En contraste con los roles como el de líder de proyecto, que podrían parecer análogos, el Scrum master no indica al equipo qué se debe hacer en

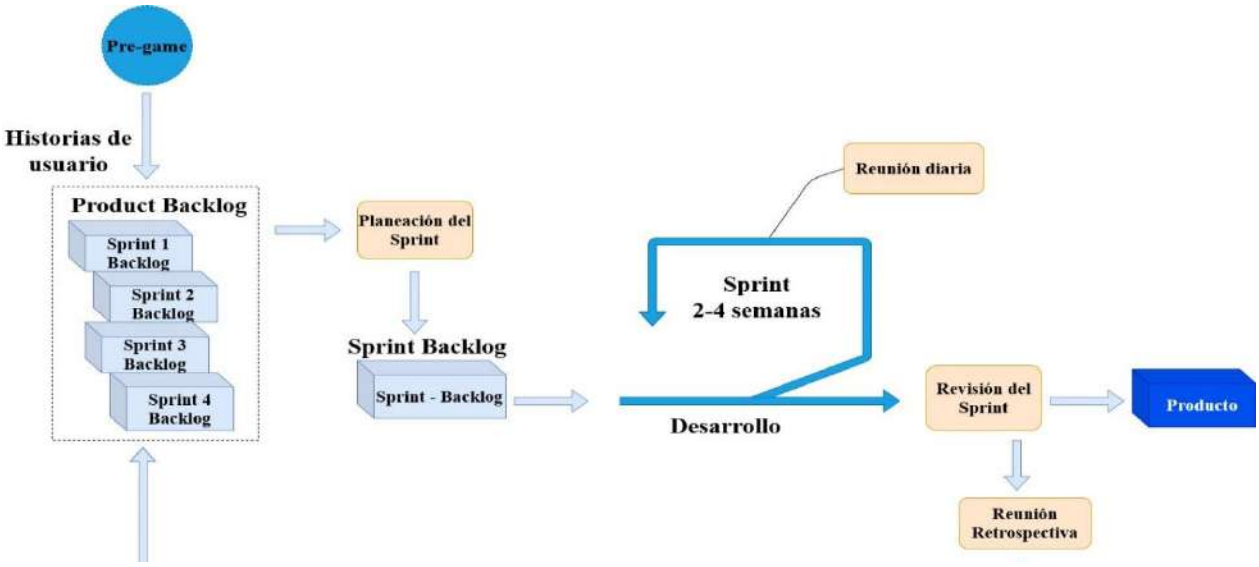


Figura 2.3. Modelo de desarrollo aplicando Scrum (González, 2008).

cada iteración. El Scrum master, sirve de ayuda al propietario y al equipo en el sentido que les ayuda a eliminar dificultades durante el proceso de desarrollo y promover el buen uso de prácticas ágiles de trabajo (Cervantes Maceda, Velasco-Elizondo, & Castro Careaga, 2016).

En la figura 2.3 se muestra el ciclo de vida del desarrollo que propone Scrum para un producto software y se describen sus componentes principales.

El proceso

Las historias de usuario son el elemento base que utiliza Scrum para describir las características que el usuario espera que tenga el software que se va a desarrollar. (Cohen, 2004). Tanto se pueden incorporar funciones del sistema como cualquier otro aspecto (restricciones, rendimiento o aspecto). Las historias de usuario se presentan desde la perspectiva del usuario, es decir, no se describen usando terminologías técnicas sino se usa un lenguaje relacionado a la aplicación que se está desarrollando para que sea comprensible por los clientes y los desarrolladores.

El proceso comienza con la fase de *Pre-game* o historias de usuario, en la que se realiza de forma conjunta entre el **producto owner** y el cliente definiendo de forma sencilla y clara las características que debe tener el software que se vaya a desarrollar. Posteriormente se refinarán cada historia de usuario para obtener las tareas necesarias para llevar a cabo el desarrollo de ellas. El resultado de la fase *Pre-game* es el *Product Backlog* o pila de producto, el cual contiene una lista de todas las historias de usuario priorizadas, así como de las tareas que se deben llevar a cabo para la realización del proyecto. La generación del *Product Backlog* se obtiene con la ayuda del cliente/usuario del software que se va a desarrollar, esto ayuda resolver dudas sobre las historias de usuarios en ese mismo instante. Una vez identificadas las historias de usuarios del *Product Backlog*, se separan en etapas de corta duración (no más de 30 días) denominadas Sprints. Para cada Sprint se realiza una reunión de planificación de lo que se realizará en ese Sprint y se estable fecha de finalización del mismo, en esta reunión está presente el **equipo de desarrollo**, el **producto**

owner y el **Scrum master**. El objetivo es mover aquellas historias de usuario con mayor prioridad para el cliente al denominado *Sprint Backlog*, que contiene el conjunto de tareas a desarrollar en ese sprint por parte del equipo de desarrollo, incluyendo su diseño, desarrollo, pruebas, integración, etc. Las historias de usuario se congelan en el *Sprint Backlog o pila de iteraciones* de forma que en ese periodo no pueden producirse cambios sobre los aspectos que están en desarrollo.

Se realizan reuniones todos los días entre el **equipo de desarrollo** y el **Scrum master** durante un sprint, las cuales duran máximo 15 minutos, en la cual sólo se comentan que tareas se realizaron y dificultades que surgieron y como solucionarlas. Al final de cada sprint se obtiene un producto entregable y esto se vuelve un pequeño incremento. En este punto se procede a una revisión del proyecto para mostrar los avances e incorporar nuevas historias de usuario al *Product Backlog*, si son requeridas por el cliente. Se establece que un incremento o entrega del producto está 100% completa si cumple con los criterios de las historias de usuarios presentadas al inicio de un Sprint (González, 2008).

Tabla 2.1. Ranking de "agilidad" (Los valores más altos representan una mayor agilidad) (Highsmith, 2002).

	CMM	ASD	Crystal	DSDM	FDD	LD	Scrum	XP
Sistema como algo cambiante	1	5	4	3	3	4	5	5
Colaboración	2	5	5	4	4	4	5	5
Características de la metodología (CM)								
- Resultados	2	5	5	4	4	4	5	5
- Simplicidad	1	4	4	3	5	3	5	5
- Adaptabilidad	2	5	5	3	3	4	4	3
- Excelencia técnica	4	3	3	4	4	4	3	4
- Prácticas de colaboración	2	5	5	4	3	3	4	5
Media CM	2.2	4.4	4.4	3.6	3.8	3.6	4.2	4.4
Media Total	1.7	4.8	4.5	3.6	3.6	3.9	4.7	4.8

En la tabla 2.1, se comparan las distintas aproximaciones ágiles en base a tres parámetros las cuales son: 1) Vista del sistema como algo cambiante, 2) Consideración de la colaboración entre los miembros del equipo y 3) Características más específicas de la propia metodología como son Simplicidad, Excelencia Técnica, Resultados, Adaptabilidad, etc. También incorpora como referencia no ágil el *Capability Madurity* Model (CMM) (Letelier & Panadés, 2012). Con respeto a los valores de la tabla 2.1 donde 1 es el valor más bajo y 5 es el más alto en la escala numérica del 1 al 5. Por ejemplo, haciendo una comparativa entre la metodología Scrum y XP tomando como referencia la característica adaptabilidad, donde Scrum tiene un valor de 4 y XP de 3, esto quiere decir que Scrum tiene mayor adaptabilidad. Para obtención de la media CM se realiza la suma de los valores de las características de la metodología y se divide entre el número de características de la metodología. La obtención de la media total es sumando la media CM con los valores de los parámetros **Sistema como algo cambiante** y **Colaboración**, el resultado de esta suma se divide entre 3, el cual corresponde a los tres parámetros mencionados anteriormente. En la tabla 2.1 se puede observar que debido a la superioridad de Scrum sobre XP respecto a la adaptabilidad está entre las metodologías con mayor agilidad.

2.2 Lenguaje Unificado de Modelado (UML)

El modelado de sistemas es el proceso para desarrollar modelos abstractos de un sistema, donde cada modelo presenta una visión o perspectiva diferente de dichos sistemas. En general, el modelado de sistemas se ha convertido en un medio para representar el sistema usando algún tipo de notación gráfica, que ahora casi siempre se basa en notaciones del UML (Sommerville, 2016).

El UML tiene numerosos tipos de diagramas y, por lo tanto, soporta la creación de diferentes tipos de modelos de sistemas. Sin embargo, un estudio en 2007 (Erickson y Siau, 2007) mostró que

la mayoría de los usuarios del UML consideraban que cinco tipos de diagrama podrían representar lo esencial de un sistema (Sommerville, 2016).

1. **Diagramas de actividad**: Muestran las actividades incluidas en un proceso o en el procesamiento de datos.
2. **Diagramas de casos de uso**: Exponen las interacciones entre un sistema y su entorno.
3. **Diagramas de secuencias**: Muestran las interacciones entre los actores y el sistema, y entre los componentes del sistema.
4. **Diagramas de clase**: Revelan las clases de objeto en el sistema y las asociaciones entre las clases.
5. **Diagramas de estado**: Explican cómo reacciona el sistema frente a eventos internos y externos.

A continuación, se fundamentan los tres tipos de diagramas que serán implementados con el fin de mostrar el modelado de la aplicación propuesta en el objetivo de la tesis.

2.2.1 Casos de uso

El modelado de casos de uso se utiliza ampliamente para apoyar la adquisición de requerimientos. Un caso de uso puede tomarse como un simple escenario que describa lo que espera el usuario de un sistema.

Cada caso de uso representa una tarea discreta que implica interacción externa con un sistema. En su forma más simple, un caso de uso se muestra como una elipse, con los actores que intervienen en el caso de uso representados como figuras humanas. La figura 2.4 presenta como ejemplo un caso de uso de un sistema de atención de pacientes a la salud mental; que implica la tarea de subir datos desde este sistema hasta un sistema más general de registro de pacientes. Este sistema más

general mantiene un resumen de datos sobre el paciente, en vez de los datos sobre cada consulta, que se registran en el sistema de administración de pacientes (Sommerville, 2016).

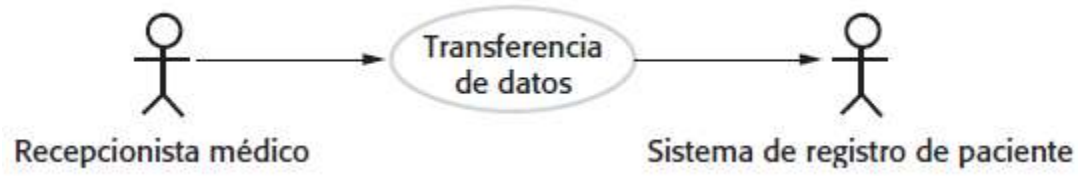


Figura 2.4. Caso de uso de transferencia de datos (Sommerville, 2016).

Se observa que en este caso de uso hay dos actores: el operador que transfiere los datos y el sistema de registro de pacientes. La notación con figura humana se desarrolló originalmente para cubrir la interacción entre individuos, pero también se usa ahora para representar otros sistemas externos y el hardware. De manera formal, los diagramas de caso de uso deben emplear líneas sin flechas; las flechas en el UML indican dirección del flujo de mensajes. Evidentemente, en un caso de uso los mensajes pasan en ambas direcciones. Sin embargo, las flechas en la figura 2.4 se usan de manera informal para indicar que el recepcionista médico inicia la transacción y los datos se transfieren al sistema de registro de pacientes.

Los diagramas de caso de uso brindan un panorama sencillo de una interacción, de modo que se tiene que ofrecer más detalle para entender lo que está implicado, el formato más útil es un formato tabular estándar. En la tabla 2.2 se muestra una descripción del caso de uso “transferencia de datos” (Sommerville, 2016).

Tabla 2.2. Descripción tabular del caso de uso "transferencia de datos" (Sommerville, 2016).

Sistema de atención de pacientes a la salud mental	
Actores	Recepcionista médico, sistema de registro de pacientes.
Descripción	Un recepcionista puede transferir datos del sistema de atención de pacientes a la salud mental a una base de datos general de registros de pacientes, mantenida por una autoridad sanitaria. La información transferida puede ser información personal actualizada o un resumen del diagnóstico y tratamiento del paciente.

Datos	Información personal del paciente, resumen de tratamiento.
Estimulo	Comando de usuario emitido por recepcionista médico.
Respuesta	Confirmación de que el sistema de registro de pacientes se actualizó.
Comentarios	El recepcionista debe tener permisos de seguridad adecuados para acceder a la información del paciente y al sistema de registro de pacientes.

En la figura 2.5 se muestra el caso de uso de las acciones que puede hacer el **recepcionista médico** con el módulo de **pacientes** tales como: **registrar**, **dar de baja**, **ver información**, **transferir datos** y **contacto del paciente**.

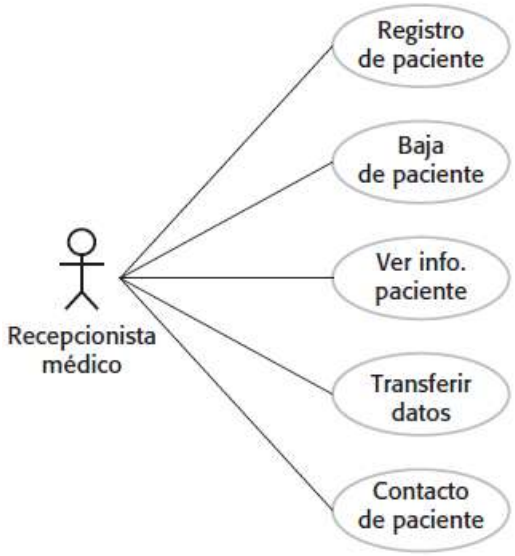


Figura 2.5. Casos de uso que involucran el papel "recepcionista médico" (Sommerville, 2016).

En la tabla 2.3 se describen las relaciones que pueden ser utilizadas entre casos de uso y actores esto se puede observar en la figura 2.5.

Tabla 2.3. Tipos de relaciones de casos de uso (Rumbaugh, Jacobson, & Booch, 2000).

Relación	Función	Notación
Asociación	La línea de comunicación entre un actor y un caso de uso en el que participa.	———

Extensión	La inserción de comportamiento adicional en un caso de uso base que no tiene conocimiento sobre él.	«extend»
Generalización	Una relación entre un caso de uso general y un caso de uso más específico, que hereda y añade propiedades a aquél.	
Inclusión	Inserción de comportamiento adicional en un caso de uso base, que describe explícitamente la inserción.	«include»

Las relaciones de inclusión y extensión se dibujan como flechas de líneas discontinuas con la palabra clave *include* o *extend* como se muestra en la figura 2.6. La relación de inclusión apunta al caso de uso a ser incluido; la relación de extensión señala el caso de uso que se extenderá.

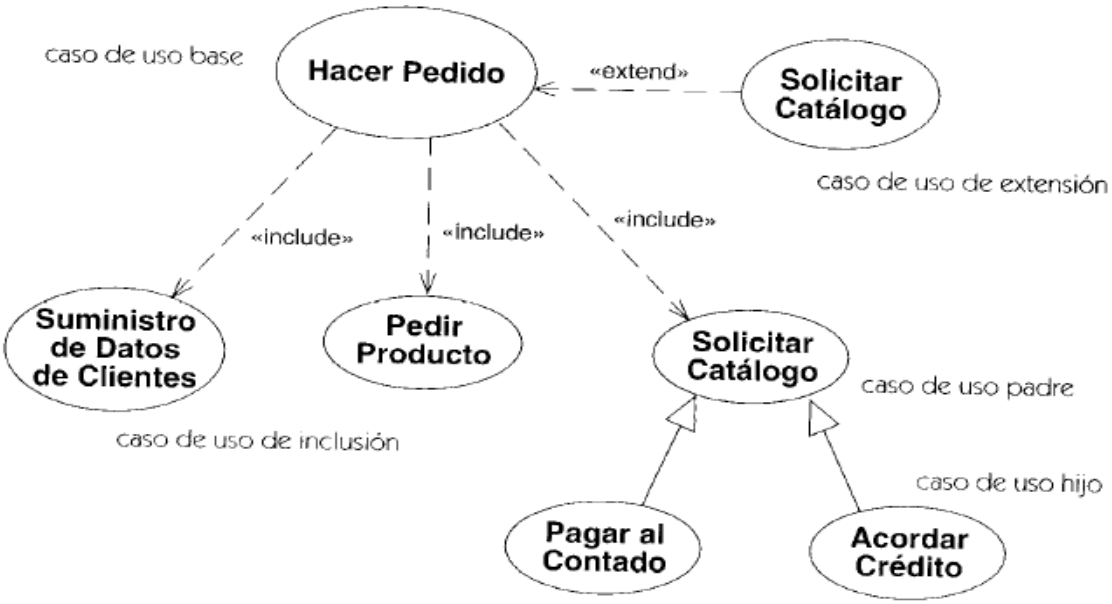


Figura 2.6. Relaciones de casos de uso (Rumbaugh, Jacobson, & Booch, 2000).

2.2.2 Diagrama de secuencia

Los diagramas de secuencia en el UML se usan principalmente para modelar las interacciones entre los actores y los objetos en un sistema, así como las interacciones entre los objetos en sí. Un diagrama de secuencia muestra la sucesión de interacciones que ocurre durante un caso de uso

particular o una instancia de caso de uso. La figura 2.7 es un ejemplo de un diagrama de secuencia que ilustra los fundamentos de la notación. Estos modelos de diagrama incluyen las interacciones en el caso de uso **ver información del paciente** (ver figura 2.5), donde un recepcionista médico puede conocer la información de algún paciente.

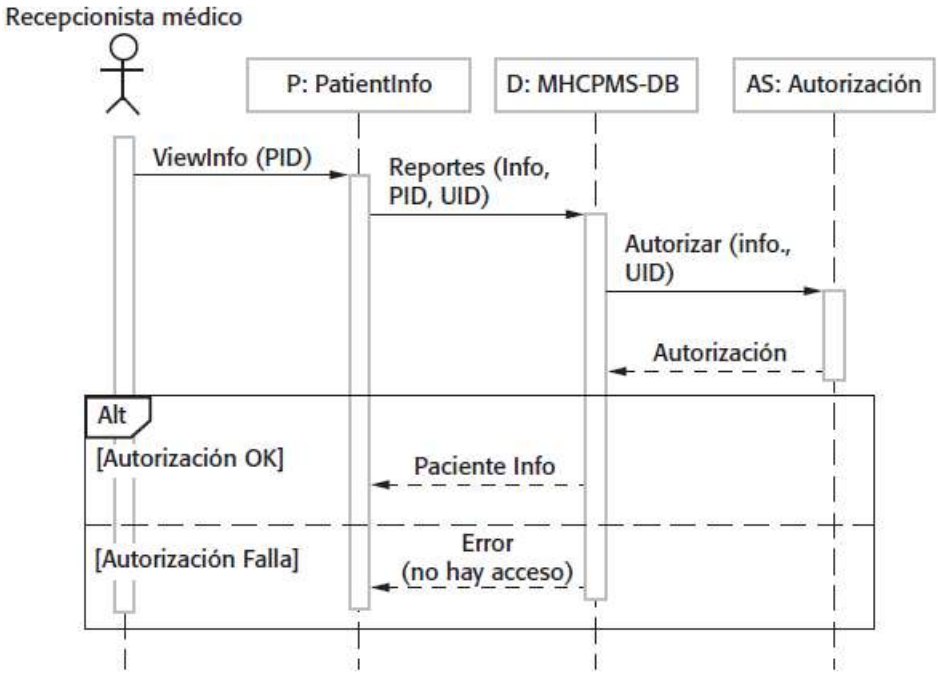


Figura 2.7. Diagrama de secuencia para ver información del paciente (Sommerville, 2016).

Los objetos y actores que intervienen se mencionan a lo largo de la parte superior del diagrama, con una línea punteada que se dibuja verticalmente a partir de éstos. Las interacciones entre los objetos se indican con flechas dirigidas. El rectángulo sobre las líneas punteadas indica la línea de vida del objeto tratado (es decir, el tiempo que la instancia del objeto está involucrada en la computación). La secuencia de interacciones se lee de arriba abajo. Las anotaciones sobre las flechas señalan las llamadas a los objetos, sus parámetros y los valores que regresan. En este ejemplo, también se muestra la notación empleada para exponer alternativas. Un recuadro marcado con **Alt** se usa con las condiciones indicadas entre corchetes.

La ilustración 2.8 se lee del siguiente modo:

1. El recepcionista médico activa el método **ViewInfo** (ver información) es una instancia **P** de la clase de objeto **PatientInfo**, y suministra el identificador del paciente, **PID**. **P** es un objeto de interfaz de usuario, que se despliega como un formato que muestra la información del paciente.
2. La instancia P llama a la base de datos para regresar la información requerida, y suministra el identificador del recepcionista para permitir la verificación de seguridad.
3. La base de datos comprueba, mediante un sistema de autorización, que el usuario esté autorizado por tal acción.
4. Si está autorizado, se regresa la información del paciente y se llena un formato en la pantalla del usuario. Si la autorización falla, entonces se regresa un mensaje de error.

2.2.3 Diagrama de clase

Los diagramas de clase pueden usarse cuando se desarrolla un modelo de sistema orientado a objetos para mostrar las clases en un sistema y las asociaciones entre dichas clases. De manera sintetizada, una **clase de objeto** se considera como una definición general de un tipo de objeto del sistema. Una **asociación** es un vínculo entre clases, que indica que hay una relación entre dichas clases. En consecuencia, cada clase puede tener algún conocimiento de esta clase asociada.

Cuando se desarrollan modelos durante las primeras etapas del proceso de ingeniería de software, los objetos representan algo en el mundo real, como un paciente, una receta, un médico, etcétera. Conforme se desarrolla la implementación, por lo general se necesitan definir los objetos de implementación adicionales que se usan para dar la funcionalidad requerida del sistema. En los

diagramas de clase, el enfoque está sobre el modelado de objetos del mundo real, como parte de los requerimientos o los primeros procesos de diseño del software.

Los diagramas de clase en el UML pueden expresarse con diferentes niveles de detalle. Cuando se desarrolla un modelo, la primera etapa con frecuencia implica identificar los objetos esenciales y representarlos como clases. La forma más sencilla de hacer esto es escribir el nombre de la clase en un recuadro. También puede anotar la existencia de una asociación dibujando simplemente una línea entre las clases. (Sommerville, 2016).

Figura 2.8. Clases y asociación UML (Sommerville, 2016).

Por ejemplo, la figura 2.8 es un diagrama de clase simple que muestra dos clases: **Paciente** y **Registro del paciente**, con una asociación entre ellos.

En la figura 2.8 se muestra una característica más de los diagramas de clase: la habilidad para mostrar cuántos objetos intervienen en la asociación. En este ejemplo, cada extremo de la asociación se registra con un 1, lo cual significa que hay una relación 1:1 entre objetos de dichas clases. Esto es, cada paciente tiene exactamente un registro, y cada registro conserva información precisa del paciente. En el siguiente diagrama se observa que son posibles otras multiplicidades. Se define un número exacto de objetos que están implicados, o bien, con el uso de un asterisco (*), como se muestra en la figura 2.9, que hay un número indefinido de objetos en la asociación.

En la figura 2.9 se desarrolla este tipo de diagrama de clase para mostrar que los objetos de la clase **Paciente** también intervienen en relaciones con varias otras clases. Asimismo, el UML permite especificar el papel de los objetos que participan en la asociación (Sommerville, 2016).

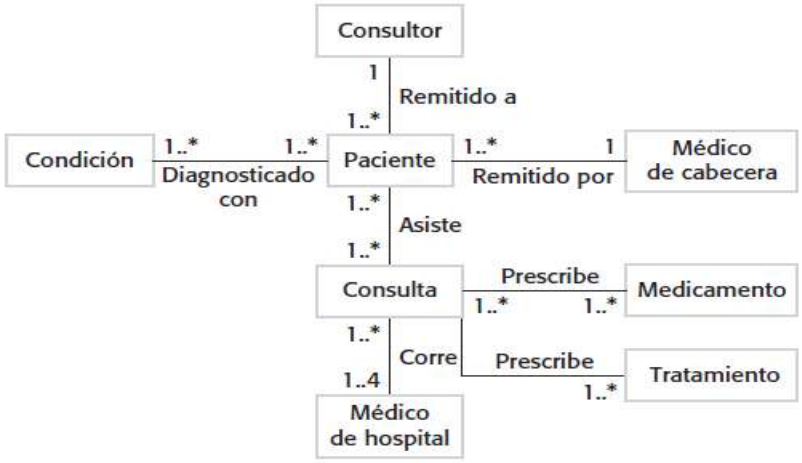


Figura 2.9. Clases y asociaciones en el sistema de atención de pacientes a la salud mental (Sommerville, 2016).

En este nivel de detalle, los diagramas de clase parecen modelos semánticos de datos. Los modelos semánticos de datos se usan en el diseño de bases de datos. Muestran las entidades de datos, sus atributos asociados y las relaciones entre dichas entidades. El UML no incluye una notación específica para este modelado de bases de datos, ya que supone un proceso de desarrollo orientado a objetos, así como modelos de datos que usan objetos y sus relaciones. Sin embargo, es posible usar el UML para representar un modelo semántico de datos. En un modelo semántico de datos, se piensa en entidades como clases de objeto simplificadas (no tienen operaciones), atributos como atributos de clase de objeto y relaciones como nombres de asociaciones entre clases de objeto. Cuando se muestran las asociaciones entre clases, es conveniente representar dichas clases en la forma más sencilla posible. Para definir con más detalle una clase, se colocan sus atributos (las características de un objeto) u operaciones (aquello que se puede solicitar de un objeto). Por ejemplo, un objeto **Paciente** tendrá el atributo **Dirección** y puede incluir una operación llamada **CambiarDireccion**, que se llama cuando un paciente manifiesta que se mudó de una dirección a otra. En el UML, los atributos y las operaciones se muestran al extender el rectángulo simple que representa una clase. Esto se muestra en la figura 2.10, donde:

1. El nombre de la clase de objeto está en la sección superior.

2. Los atributos de la clase están en la sección media. Esto debe incluir los nombres del atributo y, opcionalmente sus tipos.
3. Las operaciones (llamadas métodos en los lenguajes de programación) asociadas con la clase de objeto, están en la sección inferior del rectángulo. Las operaciones pueden contener opcionalmente parámetros y tipo devuelto (Sommerville, 2016).

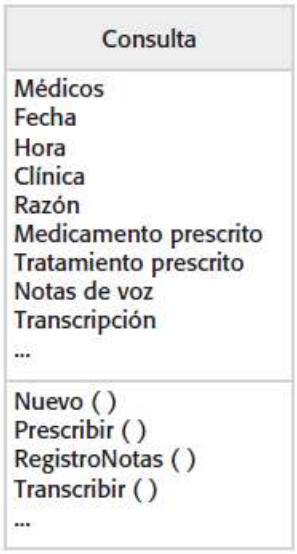


Figura 2.10. La clase de consulta (Sommerville, 2016).

Una vez descritos los tipos de diagramas a emplear en este trabajo, es recomendable tener en consideración que en el desarrollo de software existe una separación entre entidades principales, el Front-End y el Back-End estos representan un tipo de abstracción que ayuda mantener separada las diferentes tecnologías y partes que integran un software. A continuación, se describen estas dos separaciones.

2.3 Separación de preocupaciones (Separation of Concerns, SoC)

Uno de los principios más importantes en la ingeniería de software es el principio de separación de preocupaciones. Este principio establece que un problema dado involucra diferentes tipos de preocupaciones, que deben identificarse y separarse para hacer frente a la complejidad y para lograr

los factores de calidad de ingeniería requeridos, como la solidez, adaptabilidad y reutilización (Aksit, Tekinerdogan, & Bergmans, 2001).

2.3.1 Front-End

Es la parte del software que se enfoca en el usuario, es decir, con lo que puede interactuar gráficamente (interfaz gráfica de usuario). Existen diferentes **frameworks**, **preprocesadores** y **librerías** que ayudan a proporcionar experiencia de usuario, inmersión y usabilidad.

2.3.2 Back-End

Es la parte que se enfoca en procesar los datos provenientes de las entradas desde el **Front-End** para que funcione correctamente el software. En términos generales, es la parte donde están las reglas de negocio y en esta parte interactúan lenguajes programación, así como frameworks y librerías que realizan este proceso.

Estas dos separaciones, en la literatura se conoce comúnmente como de **lado del cliente** (Front-End) y de **lado del servidor** (Back-End), más adelante se describirán algunas herramientas de desarrollo que actúan en estas dos separaciones.

Un aspecto muy importante en el desarrollo de software es la arquitectura con la cual se trabajará, esta define una estructura de cómo debe ser construido el software, a continuación, se describen algunas de las arquitecturas que son utilizadas para construcción de software.

2.4 Arquitectura de software

La arquitectura de software se interesa por entender cómo debe organizarse un sistema y cómo tiene que diseñarse la estructura global de ese sistema. En el modelo del proceso de desarrollo de software, la arquitectura de software es la primera etapa en el proceso de diseño del software. Es

el enlace crucial entre el diseño y la ingeniería de requerimientos, ya que identifica los principales componentes estructurales en un sistema y la relación entre ellos.

La arquitectura de software es importante porque afecta el desempeño y la potencia, así como la capacidad de distribución y mantenimiento de un sistema (Bosch, 2000). Como afirma Bosch, los componentes individuales implementan los requerimientos funcionales del sistema. Los requerimientos no funcionales dependen de la arquitectura del sistema, es decir, la forma en que dichos componentes se organizan y comunican. En muchos sistemas, los requerimientos no funcionales están también influidos por componentes individuales, pero no hay duda de que la arquitectura del sistema es la influencia dominante (Sommerville, 2016).

Existen diferentes arquitecturas de software que proponen una manera diferente de organizar un sistema de software, las cuales se describirán en los siguientes apartados.

2.4.1 Arquitectura en capas

Organiza un sistema en capas donde las funcionalidades están relacionadas con cada capa. Una capa proporciona servicios a la capa superior, las capas de nivel más bajo representan funcionalidades o servicios centrales que pueden ser usados en todo el sistema, como se muestra en la figura 2.11 (Sommerville, 2016).

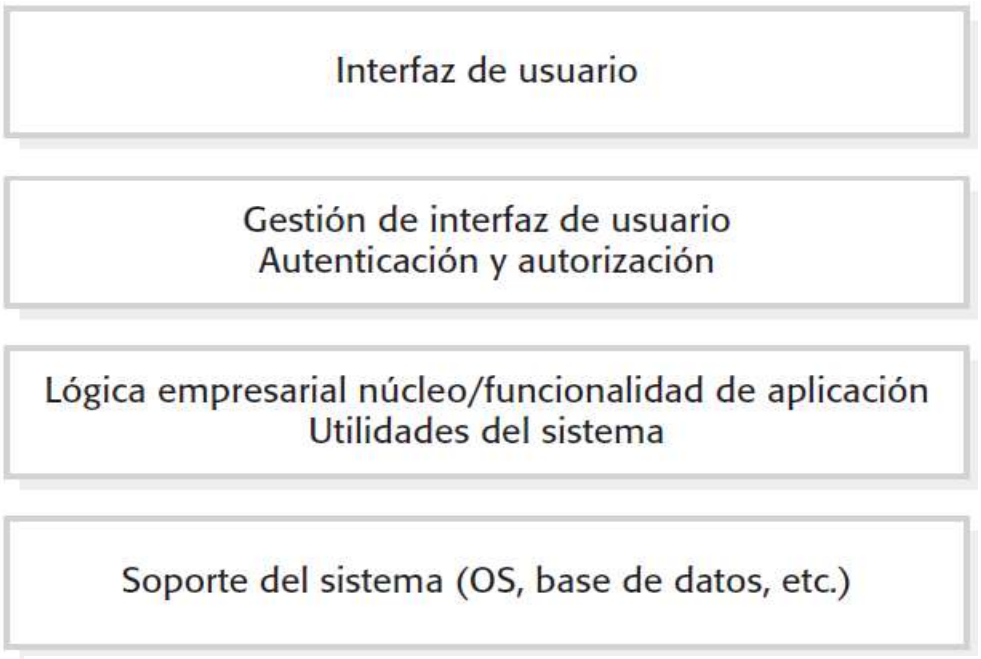


Figura 2.11. Arquitectura genérica en capas (Sommerville, 2016).

Un ejemplo para un modelo en capas de un sistema para compartir documentos con derechos de autor se tiene en diferentes bibliotecas, como se muestra en la figura 2.12.

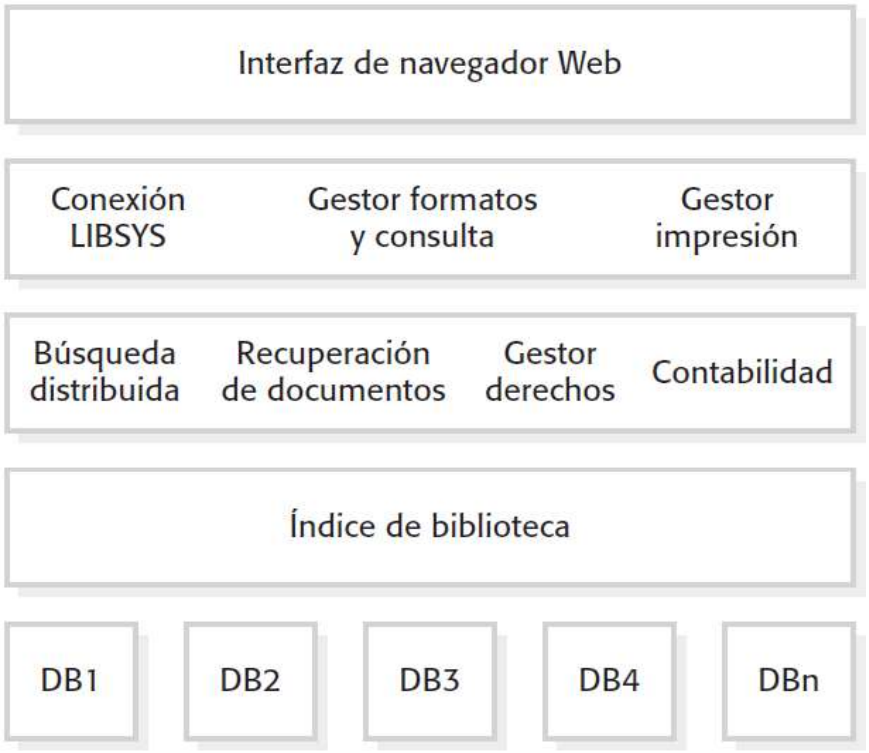


Figura 2.12. Arquitectura del sistema de biblioteca llamado LIBSYS (Sommerville, 2016).

2.4.2 Arquitectura de repositorio

Todos los datos en un sistema se gestionan en un repositorio central, accesible a todos los componentes del sistema. Los componentes no interactúan directamente, sino tan sólo a través del repositorio. En la figura 2.13 se muestra un ejemplo de un **IDE** o **Entorno de Desarrollo Integrado (en inglés, Integrated Development Environment)** donde los componentes usan un repositorio de información de diseño de sistema. Cada herramienta de software genera información que, en ese momento, está disponible para uso de otras herramientas (Sommerville, 2016).

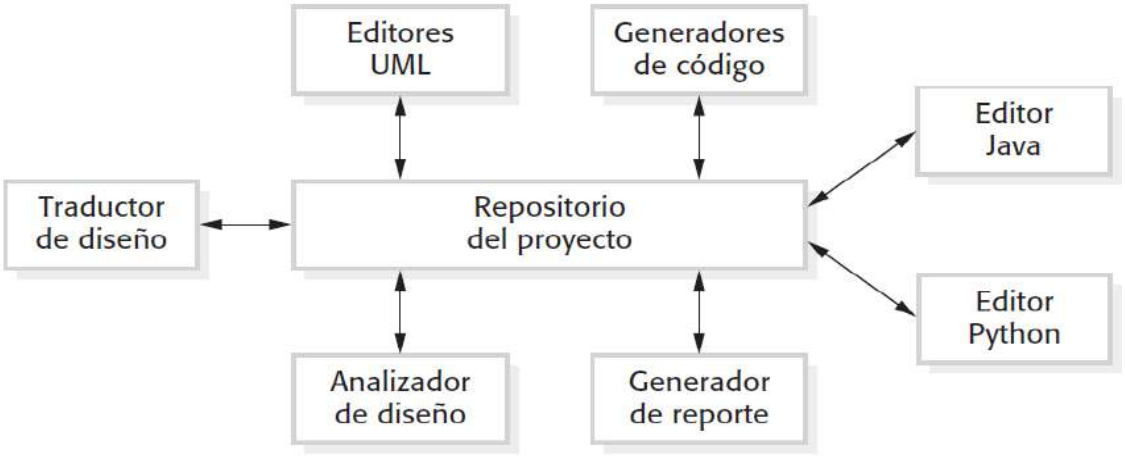


Figura 2.13. Arquitectura de repositorio para un IDE (Sommerville, 2016).

2.4.3 Arquitectura cliente-servidor

En una arquitectura cliente-servidor, la funcionalidad del sistema se organiza en servicios, y cada servicio lo entrega un servidor independiente. Los clientes son usuarios de dichos servicios y para utilizarlos ingresan a los servidores. En la figura 2.14 se muestra de una filmoteca y videoteca (videos/DVD) organizada como un sistema cliente-servidor (Sommerville, 2016).

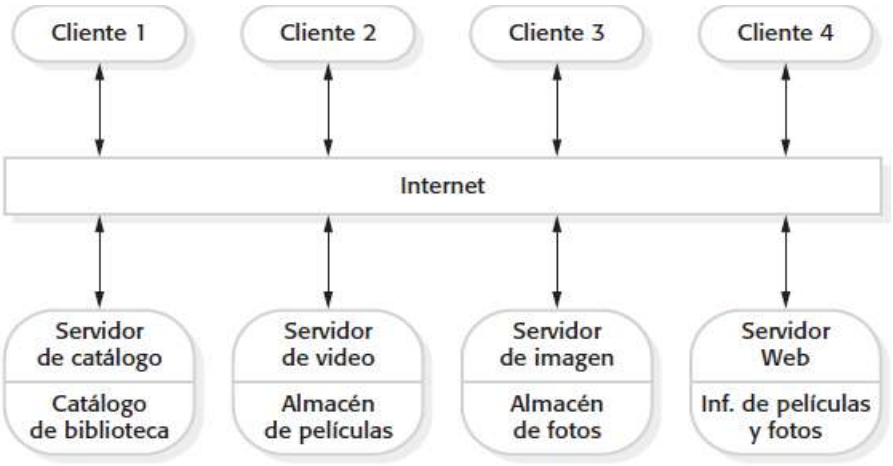


Figura 2.14. Arquitectura cliente-servidor para una filmoteca (Sommerville, 2016).

2.4.4 Arquitectura de tubería y filtro (pipe and filter)

El procesamiento de datos en un sistema se organiza de forma que cada componente de procesamiento (filtro) sea discreto y realice un tipo de transformación de datos. Los datos fluyen (como en una tubería) de un componente a otro para su procesamiento. Un ejemplo de un sistema de tubería y filtrado usado para el procesamiento de facturas como se muestra en la figura 2.15 (Sommerville, 2016).

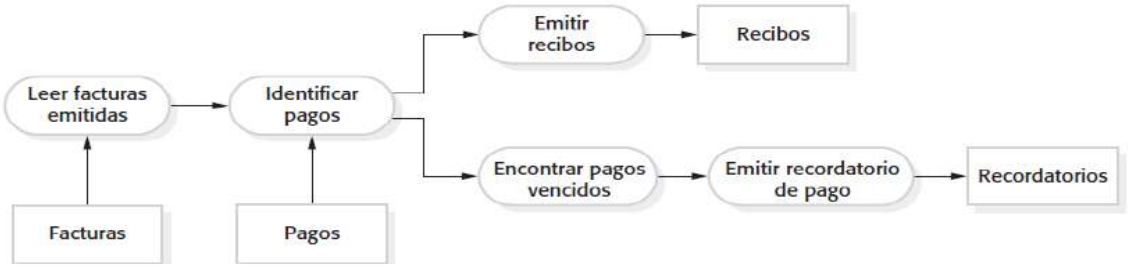


Figura 2.15. Arquitectura de tubería y filtro (Sommerville, 2016).

2.4.5 Arquitectura de tres capas

El patrón multicapa descompone una aplicación mono capa en varias capas. El objetivo principal es separar los componentes de acuerdo a su función, por ejemplo, en las aplicaciones hay

componentes encargados de la presentación, otros de la lógica de negocio y otros de la persistencia de datos.

En ocasiones se confunde el término “capas” con el término “nivel”. El primer término se utiliza para referenciar a las distintas “partes” en las que una aplicación se divide desde el punto de vista lógico. Mientras que el segundo término corresponde a la forma física en que una aplicación se organiza. Un ejemplo muy simple (descrito mediante la figura 2.16) y que es muy común encontrar, es una aplicación que tiene dos niveles (nivel de aplicación y nivel de datos), en donde cada nivel puede tener varias capas. En este caso, el nivel de aplicación puede estar constituido por la capa de presentación y por la capa de lógica de negocio y el nivel de datos puede contener sólo la capa de persistencia de datos, lo cual da como resultado una arquitectura de tres capas. En el nivel de aplicación las capas interactúan entre sí por medio de las interfaces implementadas en la capa de lógica de negocio.

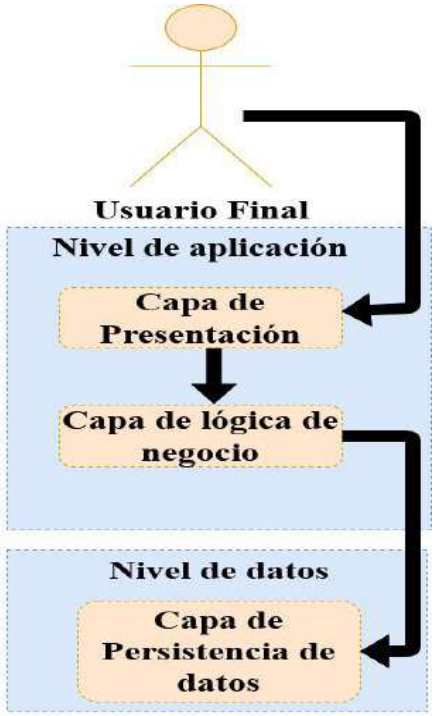


Figura 2.16. Arquitectura dividida en dos niveles y tres capas (Tahuiton Mora, 2011).

Estas interfaces permiten a la capa de lógica de negocio proveer los recursos que necesita la capa de presentación. Las capas inferiores se encargarán de brindar sus servicios a las capas superiores por medio de sus interfaces.

Al separar una aplicación en capas y niveles se tiene la capacidad de modificar de forma independiente cada capa (Tahuiton Mora, 2011).

Para el desarrollo del producto de esta tesis, se utiliza el patrón **Modelo-Vista-Controlador (MVC)** ya que es adaptable para el desarrollo de aplicaciones web. Por tal motivo, es la arquitectura seleccionada, ya que la aplicación que se pretende desarrollar, está basada en web. Además, la mayoría de los frameworks existentes para la construcción de aplicaciones web de lado del Front-End y Back-End implementan este patrón arquitectónico debido a la separación de intereses o conceptos dentro de una aplicación, ayudando a que exista un orden en la codificación, facilitando el mantenimiento y como gran parte de los patrones de diseño que estos permiten la reutilización del código.

2.4.6 Modelo-Vista-Controlador o MVC (Model-View-Controller)

Este modelo separa la presentación e interacción de los datos del sistema. El sistema se estructura en tres componentes lógicos que interactúan entre sí. El componente **modelo** maneja los datos del sistema y las operaciones asociadas a esos datos. El componente **vista** define y gestiona cómo se presentan los datos al usuario. El componente **controlador** dirige la interacción del usuario y pasa estas interacciones a la **vista** o **modelo** (Sommerville, 2016). En la figura 2.17 se ilustra la interacción y organización de los componentes de MVC.

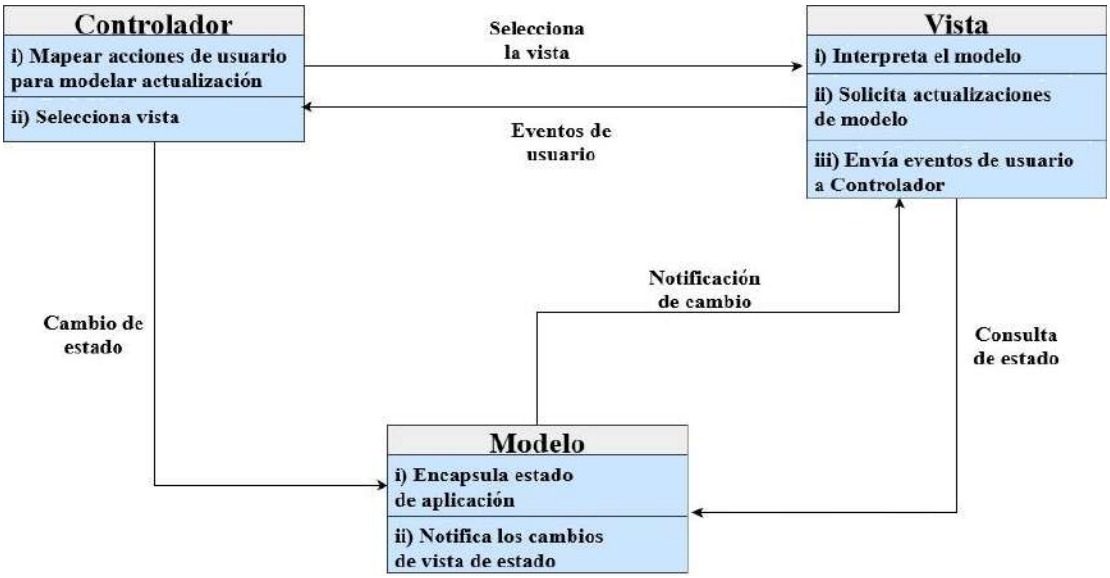


Figura 2.17. La organización del MVC (Sommerville, 2016).

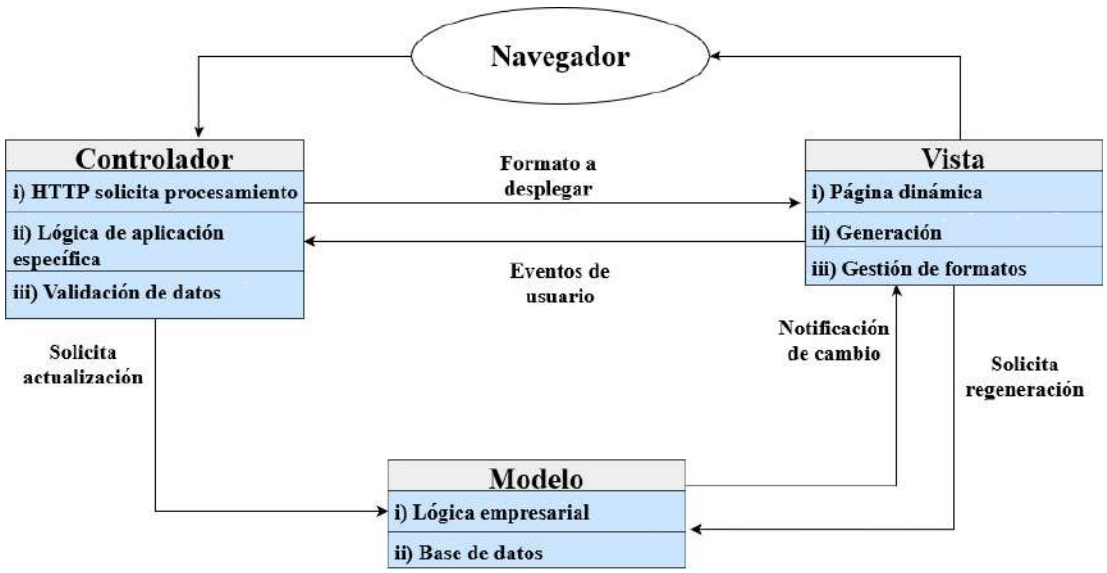


Figura 2.18. Arquitectura de aplicación web con el patrón MVC (Sommerville, 2016).

En la figura 2.18 se muestra un ejemplo la arquitectura de un sistema de aplicación basado en la Web, que se organiza con el uso del patrón MVC.

Después de haber documentado las diferentes arquitecturas de software, en la siguiente sección se describirán los lenguajes de programación y frameworks que actúan en el Front-End y posteriormente las que actúan en el Back-End, con los cuales se realizará la construcción de la aplicación propuesta en el objetivo de la tesis.

2.5 Herramientas de desarrollo

Una aplicación web está compuesta por diferentes tecnologías, pero todas comparten una misma tecnología que es HTML y esto es debido a que el navegador sólo interpreta documentos en formato HTML.

2.5.1 Lenguaje de marcas de hipertexto o HTML (HyperText Markup Languaje)

Es un lenguaje utilizado para definir la presentación de información en páginas Web. Gracias a HTML se ha podido combinar texto y gráficos en una misma página y crear sistemas de presentación complejos con **hiperenlaces** entre páginas. Pero HTML no es útil en lo que se refiere a la descripción de información. Por ejemplo, se puede utilizar HTML para dar formato a una tabla, pero no para describir los elementos de datos que componen la misma (Ceballos, 2007) .

Para dar una mayor presentación a las páginas web que forman a una aplicación basada en web, se necesita establecer estilos utilizando **Hojas de estilo en cascada** (CSS) debido a que en ocasiones los estilos predeterminados en las etiquetas que proporciona HTML no son suficientes.

2.5.2 Hojas de estilo en cascada o CSS (Cascading Style Sheets)

Con CSS, se puede aplicar configuración a las páginas web para dar estilos específicos. Esto funciona porque CSS está conectado al Modelo de Objetos de Documento (DOM), con CSS se puede restaurar cualquier elemento rápidamente y fácilmente. Por ejemplo, si se desea tener el

aspecto predeterminado de una etiqueta HTML con CSS se puede asignar nuevos estilos para anular la configuración predeterminada. Una forma de agregar estilo a una página web es insertando las declaraciones necesarias usando las etiquetas **<style>** y **</style>** en el encabezado de la página, entre las etiquetas **<head>** y **</head>**. Con la aparición del estándar CSS3 en los últimos años, CSS ahora ofrece un nivel de interactividad dinámica que antes sólo era compatible con JavaScript. Por ejemplo, no sólo puede aplicar un estilo a cualquier elemento HTML para cambiar sus dimensiones, colores, bordes, espaciado, etc. Ahora también puede agregar transiciones y transformaciones animadas a sus páginas web, utilizando sólo unas pocas líneas de CSS (Nixon, 2012).

En ocasiones no basta con sólo dar una estructura y un estilo a las páginas web que componen una aplicación, ya que existen formularios para entrada de datos y estos datos tiene que ser validados o procesados antes de ser enviados al servidor y JavaScript permite realizar este tipo de acción.

2.5.3 JavaScript

JavaScript es un lenguaje de script del lado del cliente que se ejecuta completamente dentro del navegador web. Para llamarlo, se tiene que colocar el código JavaScript al abrir las etiquetas HTML **<script>** y cerrar **</script>.** JavaScript fue creado para permitir el acceso a todos los elementos de un documento HTML. En otras palabras, proporciona un medio para la interacción dinámica del usuario, como verificar la validez de la dirección de correo electrónico en los formularios de entrada. Combinado con CSS, JavaScript es el poder detrás de las **páginas web** dinámicas (Nixon, 2012).

Cuando se desarrolla una **aplicación web** en ocasiones es necesario almacenar información en el navegador para poder realizar operaciones o validaciones en el **F**ront-**E**nd, con el uso del almacenamiento web ó **web Storage** es posible hacer lo mencionado.

2.5.4 Almacenamiento web o web Storage

Con el almacenamiento web, las aplicaciones pueden almacenar datos localmente dentro del navegador del usuario. Antes de HTML5, los datos de la aplicación debían almacenarse en ***cookies***, incluidas en cada solicitud del servidor. El almacenamiento web es más seguro y se pueden almacenar grandes cantidades de datos localmente, sin afectar el rendimiento del sitio web.

A diferencia de las ***cookies***, el límite de almacenamiento es mucho mayor (al menos 5 MB) y la información nunca se transfiere al servidor. El almacenamiento web es por origen (por dominio y protocolo). Todas las páginas, de un origen, pueden almacenar y acceder a los mismos datos. El almacenamiento web HTML proporciona dos objetos para almacenar datos en el cliente:

- Window.localStorage: Almacena datos sin fecha de vencimiento.
- Window.sessionStorage: Almacena datos para una sesión (los datos se pierden cuando se cierra la pestaña del navegador) (Refsnes Data, 2020).

Con CSS se mencionó con anterioridad que proporciona diferentes estilos, pero crear estilos en ocasiones puede ser muy tardado, entonces para poder dar estilos a de una manera más fácil y rápida, actualmente existen diferentes frameworks que dan estilos a las **páginas web** con facilidad y rapidez, entre los frameworks existentes está bootstrap. A continuación, se describe este framework.

2.5.5 Bootstrap

Bootstrap, es un framework que permite crear interfaces Web con CSS, JavaScript y JQuery, cuya particularidad es adaptar la interfaz de una aplicación Web al tamaño del dispositivo en que

se visualice. Esta técnica se conoce como “responsive design” o diseño adaptativo. Los diseños creados con bootstrap son simples, limpios e intuitivos, esto da agilidad a la hora de cargar y al adaptarse a otros dispositivos (Twitter, 2011).

Para realizar funciones como la manipulación del Modelo de Objetos del Documento (DOM) o validaciones en formularios con JavaScript estas acciones conllevan en ocasiones un tiempo considerable. Para resolver este problema se utilizó JQuery.

2.5.6 JQuery

JQuery es una biblioteca de JavaScript que contiene funciones que permiten la manipulación del DOM de los documentos HTML, el manejo de eventos, animación y AJAX sean mucho más simples de usar mediante la implementación de una API permitiendo que funcione en diferentes navegadores. Con la combinación de versatilidad y extensibilidad puede ser utilizada por cualquier lenguaje de programación que actué en el Back-End (Resig, 2006).

En ocasiones es necesario enviar información de los formularios al servidor sin necesidad de refrescar la página web para que no cargue por completo todos los elementos que compone una **página web** donde JavaScript asíncrono y XML (AJAX) resuelve este problema para él envío de información.

2.5.7 JavaScript asíncrono y XML o AJAX (Asynchronous JavaScript and XML)

Esta tecnología se ejecuta de lado del cliente, es decir, en el navegador los usuarios pueden mantener la comunicación asíncrona con el servidor en segundo plano, en el sentido de que los datos adicionales se solicitan al servidor y se cargan en segundo plano sin interferir con la visualización ni el comportamiento de la página (Galeano Arenas, 2016). AJAX permite enviar y recibir información mediante los métodos POST y GET del protocolo HTTP.

En ocasiones es necesario mostrar resultados sencillos y de fácil lectura en las aplicaciones, para ello, se utilizan gráficas que representen información de manera agradable y entendible. Hoy en día existen diferentes frameworks que permiten crear diferentes tipos de gráficas para la visualización de la información en ellas, **charts js** es un framework que permite esta función.

2.5.8 Chart js

Charts js es una librería de gráficos de JavaScript que contiene una colección de funciones que ayudan a la creación de gráficos para el elemento de lienzo HTML5 o SVG. Esta biblioteca ayuda en el desarrollo y visualización de elementos gráficos como partículas, movimientos, animación, trazado, SVG y gráficos en 3D.

Esta librería se utilizará para la graficación del artefacto que genera la metodología Scrum el cual es el **burndown chart** y a su vez utilizando charts js permitirá mostrarse en los reportes generados por la herramienta propuesta.

Para el desarrollo de aplicaciones existen una gran variedad de lenguajes de programación de lado del servidor tales como java, Python, php, ruby, c++, c#, etc. Cada uno de estos lenguajes ofrecen diferentes formas de desarrollar aplicaciones, algunos de estos más complejos que otros, además de que algunos sólo están enfocados en un área en específico del desarrollo de software, pero con ciertos lenguajes se puede desarrollar aplicaciones Web, móviles y de escritorio además de ser multiplataforma, es decir que se pueden ejecutar en diferentes sistemas operativos. Después de haber descrito los lenguajes y frameworks que estarán involucrados en el Front-End, en los siguientes apartados se describirán los que estarán involucrados en el Back-End.

2.5.9 C#

Para entender lo que es C# es imprescindible decir antes lo que es Microsoft .NET Framework o .NET. Se trata de un entorno de desarrollo multilenguaje diseñado por Microsoft para simplificar la construcción, distribución y ejecución de aplicaciones para internet. Tiene fundamentalmente tres componentes: una máquina virtual (CLR: Common Language Runtime) que procesa código escrito en un leguaje intermedio (MSIL: Microsoft Intermediate Language), una biblioteca de clases (biblioteca .NET Framework) y ASP.NET que proporciona los servicios necesarios para crear aplicaciones web (Ceballos, 2012).

Precisamente C# es uno de los lenguajes de programación de alto nivel que pertenecen al paquete .NET (otros lenguajes son Visual Basic, C/C++, etc.). C# es una evolución de C/C++. Con él se pueden escribir tanto programas convencionales como para internet. Las aplicaciones podrán mostrar una interfaz gráfica de usuario o bien una interfaz de texto, como hacen las denominadas aplicaciones de consola (Ceballos, 2012).

Para el desarrollo del presente proyecto se utiliza C# ya que la aplicación a desarrollar está basada en web y este lenguaje permite también desarrollar aplicaciones web con la tecnología ASP.NET CORE el cual implementa el patrón de diseño MVC que facilita el desarrollo y mantenimiento de las aplicaciones, otras ventajas que ofrece C# es que es transportable, está orientado a objetos, el tamaño de las aplicaciones desarrollas con este lenguaje es pequeño, es decir, los archivos que genera su peso es pequeño lo cual favorece al desarrollo además de ser potente y flexible.

2.5.10 ASP.NET Core MVC

ASP.NET Core MVC es un marco de desarrollo de aplicaciones web de Microsoft que combina la efectividad y el orden de la arquitectura, las ideas y las técnicas del Modelo-Vista-Controlador (MVC) del desarrollo ágil y las mejores partes de la plataforma .NET. ASP.NET Core se basa en

.NET Core, que es una versión multiplataforma de .NET Framework sin las interfaces de programación de aplicaciones (API) específicas de Windows. Windows sigue siendo un sistema operativo dominante, pero las aplicaciones web se alojan cada vez más en contenedores pequeños y simples en plataformas en la nube, y al adoptar un enfoque multiplataforma, Microsoft ha ampliado el alcance de .NET, ha hecho posible implementar aplicaciones ASP.NET Core en un conjunto más amplio de entornos de alojamiento y, como beneficio adicional, ha hecho posible que los desarrolladores creen aplicaciones web en Linux y Mac OS/X (Freeman, 2016).

Desarrollar el proyecto con ASP.NET Core facilita su desarrollo y esto no implica que el desarrollo sea de mala calidad, sino que esta tecnología es simple y más fácil de trabajar, lo cual permite un desarrollo ágil de aplicaciones, además como se mencionó anteriormente, permite desarrollar aplicaciones multiplataforma, incluye una forma más simple de desarrollar contenido complejo. Por último, desarrollar aplicaciones web implica la combinación de varias tecnologías (por ejemplo, HTML, CSS o gestores de base de datos) y comúnmente están divididas en diferentes niveles o capas y esto hace que se combine naturalmente a los conceptos del patrón MVC.

Para la gestión y acceso a datos entre una aplicación y el gestor de base de datos se necesita establecer una comunicación, actualmente existen ciertas tecnologías que permiten esta comunicación, pero posteriormente se tiene que realizar una conversión entre tipos de sistemas incompatibles usando lenguajes de programación orientados a objetos y todo esto lo tiene que realizar el programador. Hoy en día para resolver este problema existen tecnologías que realizan el mapeo objeto-relacional (en inglés Object-relational mapping o ORM) una de estas tecnologías es Entity Framework Core desarrollada por Microsoft.

2.5.11 Entity Framework Core

Entity Framework Core, también conocido como EF Core, es un paquete de ORM producido por Microsoft que permite que las aplicaciones .NET Core almacenen datos en bases de datos relacionales. Entity Framework Core tiene una tarea clave: almacenar objetos .NET en una base de datos y recuperarlos más tarde. Dicho de otra manera, EF Core actúa como el puente entre una aplicación MVC de ASP.NET Core y una base de datos, como se muestra en la figura 2.19.

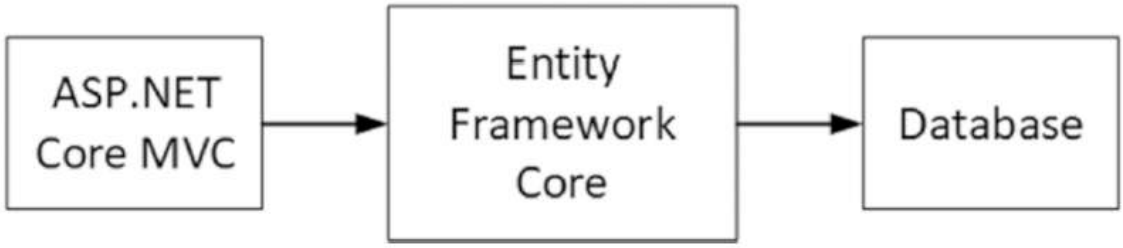


Figura 2.19. Entity Framework en contexto.

Entity Framework Core proporciona acceso fácil a la base de datos usando un lenguaje de Consulta de Idioma Integrado (en inglés, Languaje Integrated Query o LINQ) que es escrito similarmente a SQL. EF Core convierte el LINQ a SQL y se ejecuta en la base de datos. Cuando el SQL es ejecutado, EF Core toma la respuesta de la consulta y convierte los resultados dentro de un modelo para facilitar el acceso dentro del código (Freeman, 2018).

EF Core proporciona tres diferentes flujos de trabajo donde se puede configurar y usar dentro de un proyecto:

- **Database First**: Este flujo es para cuando se tiene una base de datos existente o se desea tener un control total sobre como la base de datos es creada y mantenida. Cuando se usa este flujo, se puede crear un archivo EDMX que almacena el esquema de datos, modelo de datos y la relación entre el esquema y los modelos en XML.

- **Model First**: Este flujo es un poco similar al *Database First* en que los modelos y las relaciones son mantenidas dentro de un archivo EDMX. En lugar de generar EDMX

automáticamente de un diseño de la base de datos, se crean manualmente los modelos y se define un inter-modelo de relaciones usando el diseñador de Visual Studio. Una vez finalizado, se tiene que indicar a EF Core que cree las tablas, columnas, llaves primarias y foráneas necesarias en la base de datos.

- **Code First**: Con Code First se puede indicar a EF Core que cree automáticamente la base de datos. O si se tiene una base de datos existente esta se puede modificar, se pueden usar las herramientas de EF Core para crear las clases iniciales de Code First. Cuando se usa Code First, Entity Framework proporciona un conjunto de herramientas que permiten realizar las Migraciones para modificar o actualizar automáticamente la base de datos cuando el modelo cambia (Munro, 2015).

Para facilitar la gestión y manipulación de datos para el proyecto se usará EF Core además usando el enfoque Code First debido a que da mayor control sobre las clases y objetos modelados para base de datos y proporciona una mayor facilidad para el control de las versiones de la base de datos de tal manera que si no se desea trabajar con una versión de la base de datos se puede regresar a versiones anteriores.

2.5.12 LINQ (Languaje Integrate Query)

LINQ es una tecnología de Microsoft para proporcionar un lenguaje de alto nivel que soporta mecanismos para consulta de todo tipo de datos. Estos tipos de datos incluyen arreglos en memoria y colecciones, base de datos y documentos XML. LINQ se trata de consultas, ya sean consultas que devuelven un conjunto de objetos coincidentes, un sólo objeto o un subconjunto de campos de un objeto o conjunto de objetos. En LINQ, este conjunto devuelto de los objetos se llaman una secuencia. La mayoría de las secuencias LINQ son de tipo **IEnumerable <T>**, donde **T** es el tipo de datos de los objetos almacenados en la secuencia (Freeman & Rattz, 2010).

El uso de la capa de acceso a datos para realizar las operaciones de **Crear**, **Consultar**, **Actualizar** y **Eliminar** (**Create**, **Read**, **Update** and **Delete** o **CRUD**) es bastante común en aplicaciones del mundo real. Un componente de acceso a datos aísla los detalles de cómo se llevan a cabo las operaciones insertar, leer, actualizar y borrar en un sistema. En ocasiones puede surgir la situación en la que se necesita realizar una operación en la misma base de datos (por ejemplo, un conjunto complejo de consultas) desde múltiples lugares. Aunque un componente de acceso a datos encapsula las operaciones básicas de CRUD, esto no ofrecerá ninguna ayuda para evitar la duplicación de estas operaciones. Por tanto, se termina escribiendo el mismo código en varios lugares.

En tal caso, se puede introducir una capa entre la capa de lógica de negocio y la capa de acceso a datos. Esta capa se encargará de encapsular las operaciones que se pueden reutilizar una y otra vez. Esta capa se construye siguiendo el patrón Repositorio.

2.5.13 Pattern Repository (Patrón de Repositorio)

El patrón de repositorio media entre la capa de acceso a datos y el resto del sistema. Además, lo hace proporcionando un acceso similar a una colección a los datos subyacentes. Una vez que se implementa el patrón de repositorio, la lógica de negocio no invocará componentes de la capa de acceso a datos directamente. En su lugar, invocará el patrón de repositorio para realizar el trabajo. Este patrón ofrece una interfaz de colección al proporcionar métodos para agregar, modificar, eliminar y recuperar objetos de dominio. Entity Framework ya implementa el patrón de repositorio. Por ejemplo, se puede agregar, modificar, eliminar y acceder a entidades desde un **DbSet** (representa la colección de todas las entidades en el contexto), que es bastante similar a una colección. Con esto ya no es necesario tratar con objetos ADO.NET como **SqlConnection** y

SqlCommand. Simplemente se usa el **DbSet** para realizar las operaciones en las entidades (Bipin, 2016).

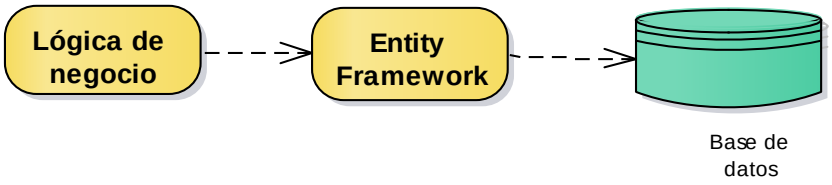


Figura 2.20. Acceso a datos sin el patrón repositorio (Bipin, 2016).

La figura 2.20 muestra la estructura de una aplicación que no implementa el patrón repositorio. La lógica de negocio invoca directamente operaciones en el Entity Framework, así como LINQ hace consultas a las entidades. Entity Framework se comunica con la base de datos subyacente para realizar el trabajo. Esta estructura se puede modificar para usar el patrón repositorio, como se muestra en la figura 2.21 (Bipin, 2016).

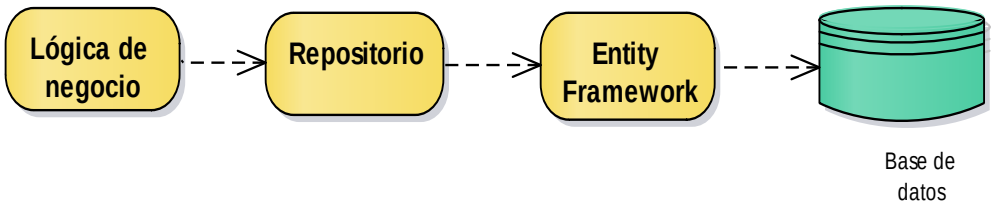


Figura 2.21. Acceso a datos con el patrón de repositorio (Bipin, 2016).

La figura 2.21 muestra un repositorio que se encuentra entre la lógica de negocio y Entity Framework. En este caso, el repositorio encapsula a través de una interfaz todas las consultas de LINQ a entidades, así como las llamadas a los métodos **ConsultaGeneral()**, **ConsultaPorId()**, **Agregar()**, **Actualizar()**, **Eliminar()** y **GuardarCambios()**, como se muestra en la figura 2.22. La capa de lógica de negocio invoca métodos del repositorio y este traduce esos métodos en las operaciones correspondientes de Entity Framework. Entity Framework se comunica con la base de datos subyacente (Bipin, 2016).

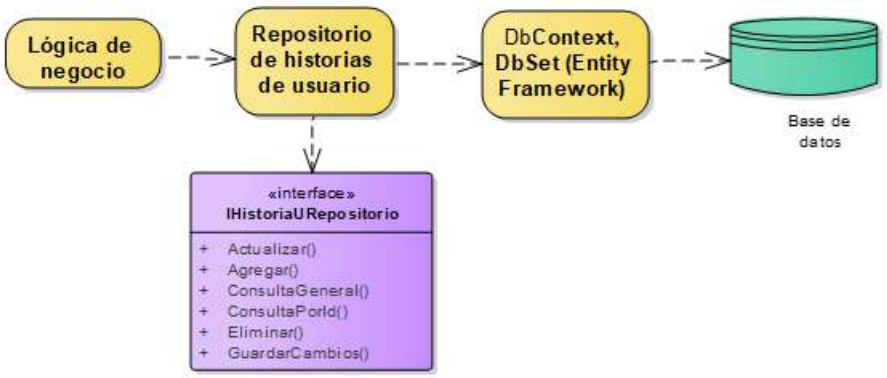


Figura 2.22. Repositorio Historias de usuario media entre la capa de lógica de negocio y Entity Framework (Bipin, 2016).

Una operación comercial puede involucrar múltiples pasos mientras es requerida como un todo, esta operación será tratada como un solo lote o unidad. Cuando se realizan transacciones en una base de datos se realizan de la siguiente manera:

- Comenzar una transacción de base de datos.
- Realizar todos los pasos de operación uno por uno contra la base de datos.
- Confirmar o deshacer la transacción.

Aunque estos pasos parezcan bastante sencillos, hay un problema: se están realizando pasos individuales de la operación directamente en la base de datos. Entonces, si una operación involucra diez pasos, se está creando diez operaciones de escritura de base de datos. Un número tan grande de pequeñas operaciones de base de datos puede afectar el rendimiento general del sistema. Es mejor capturar pasos individuales y luego enviarlos al motor de la base de datos de una vez y ejecutarlos en una sola transacción. Esa es la idea detrás del patrón Unit of Work (Bipin, 2016).

2.5.14 Pattern Unit of Work (Unidad de Trabajo)

El patrón **Unit of Work** realiza un seguimiento de una transacción comercial que se supone que debe alterar la base de datos de alguna manera. Una vez finalizada la transacción comercial, los pasos seguidos se reproducen en la base de datos en una transacción para que la base de datos refleje los cambios deseados. Por lo tanto, **Unit of Work** sigue una transacción comercial y la

convierte en una transacción de base de datos, en la que los pasos se ejecutan colectivamente como una sola unidad (Bipin, 2016).

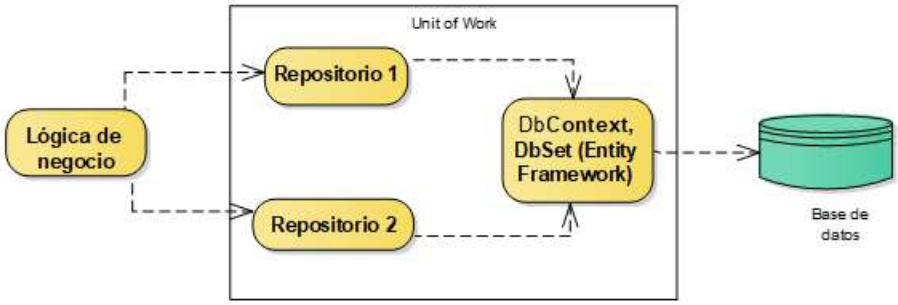


Figura 2.23. Implementación del patrón Unit of Work (Bipin, 2016).

En la figura 2.23 se muestran dos repositorios que comparten el mismo objeto DbContext (conexión a la base de datos) y objetos DbSet. Los cambios realizados por Repositorio 1 y Repositorio 2 se hacen bajo un único DbContext y por lo tanto forman un lote. Además, cuando se invoca al método **GuardarCambios**, una sola transacción de base de datos ejecuta todo el lote, convirtiéndolo en una sola unidad de trabajo (Bipin, 2016).

Actualmente gran parte de los lenguajes de programación cuentan con Entorno de Desarrollo Integrado (en inglés Integrated Development Environment o IDE) aunque también existen algunos IDE donde se pueden trabajar con diferentes lenguajes de programación. Visual Studio es un IDE el cual permite trabajar con diferentes lenguajes de programación y tecnologías, en conclusión las tecnologías mencionadas anteriormente pueden ser usadas con este IDE.

2.5.15 Visual Studio

Visual Studio es un conjunto completo de herramientas de desarrollo para construir aplicaciones web, servicios web, aplicaciones Windows o de escritorio y aplicaciones para dispositivos móviles. El entorno de desarrollo integrado que ofrece esta plataforma con todas sus herramientas y con la biblioteca de clases .NET Framework es compartido en su totalidad por Visual C#, Visual Basic y

Visual C++, permitiendo así crear con facilidad soluciones en las que intervengan varios lenguajes y en las que el diseño se realiza separadamente respecto a la programación (Ceballos Sierra, 2013).

Para la persistencia de datos se usará **PostgreSQL**, el motivo por lo que se eligió este manejador debido a que su motor no requiere tanto espacio en el servidor y no ocupa tanto procesamiento como MySQL o SQL server.

2.5.16 PostgreSQL

Es un Sistema Manejador de Base de Datos (en inglés *Database management system* o DBMS) que incorpora el modelo relacional para sus bases de datos y es compatible con el lenguaje de consulta estándar de SQL. PostgreSQL también es muy capaz y muy confiable, y tiene buenas características de rendimiento. Se ejecuta en casi cualquier plataforma UNIX, incluidos los sistemas similares a UNIX, como FreeBSD, Linux y Mac OS/X. También puede ejecutarse en servidores Microsoft Windows 2012, 2016 y 2019, o incluso en Windows 8, 8.1 y 10 para el desarrollo. Además, es de código abierto y de uso libre.

PostgreSQL puede compararse favorablemente con otros DBMS. Contiene casi todas las características que se encontraría en otros gestores de bases de datos comerciales o de código abierto, y algunas extras que tal vez no encuentre en otros productos (Matthew & Stones, 2005). Las características y funciones de PostgreSQL incluyen lo siguiente:

- Transacciones
- Subconsultas
- Vistas
- Clave foránea de integridad referencial
- Bloqueo sofisticado
- Tipos definidos por usuario

- Herencia
- Reglas
- Control de concurrencia de versiones múltiples
- Disparadores
- Restricciones (constraints).

El análisis y el diseño basado en la metodología Scrum es descrita con mayor detalle en el próximo capítulo donde se plasma la arquitectura lógica que contendrá la herramienta propuesta, además se presentan diagramas generados a partir del análisis realizado. Que darán soporte al desarrollo de la herramienta.

Capítulo 3 Análisis y diseño

Como se mencionó en el capítulo anterior, la metodología a implementar para el desarrollo de la plataforma: **Scrum**, fomenta la priorización de las historias de usuario, así como Sprints cortos, permitiendo obtener incrementos en un corto periodo. Al ser una metodología adaptativa presenta una gran flexibilidad al momento de realizar cambios en la planeación del proyecto, en este apartado de análisis y diseño se presentarán diagramas y modelos que no son propios de la metodología propuesta, sin embargo, permiten presentar una mejor comprensión de las historias de usuario.

El apartado de análisis inicia con un diagrama de modelado de negocios de la metodología Scrum, remarcando los procesos en los cuales interviene el trabajo a desarrollar en esta tesis, se han modelado diagramas específicos de estos procesos a partir de los cuales es posible crear modelados de casos de uso para los procesos remarcados.

El apartado de diseño inicia con una visión general de alto nivel de la arquitectura lógica de la plataforma, en el que es posible identificar cada uno de los niveles que contendrá la estructura, identificando en qué nivel serán implementados los patrones de unidad de trabajo y repositorio.

A partir del análisis realizado en la primera parte de este capítulo, este proporciona información para modelar diagramas de clases, que darán soporte en la implementación del presente trabajo. Concluyendo con este capítulo con un diagrama de despliegue en el que se visualizan los componentes físicos y de software necesarios para que la plataforma pueda ser implantada y utilizada.

3.1 Análisis

Como se mencionó en el capítulo 2 para el desarrollo de la plataforma se implementará la metodología Scrum y el primer proceso o etapa que se realiza es la obtención de las historias de usuario (requerimientos del sistema) para ello en el siguiente apartado se mostrará una lista de las historias de usuario para el desarrollo de la herramienta propuesta.

3.1.1 Historias de usuario

Cuando **se desarrolla** un proyecto de software implementando la metodología Scrum, el primer paso es la obtención de una lista de **historias de usuario** como se mencionó anteriormente, esta lista tiene como nombre **product backlog**. A continuación, se describen las historias de usuario obtenidas por el **product owner**, que definen con un nivel de abstracción mayor los requerimientos del sistema a construir.

Product backlog

- HU 1: La aplicación solo permitirá la creación, gestión y monitorización de los proyectos siempre y cuando el usuario se registre en la plataforma.
- HU 2: Para validar el registro de un usuario la aplicación debe enviar un correo electrónico a la dirección registrada por él.
- HU 3: El usuario tendrá el rol de **product owner** siempre y cuando haya creado el proyecto que será gestionado y monitorizado.
- HU 4: Únicamente el **product owner** podrá agregar **colaboradores**, así como podrá asignarle el rol en los proyectos.
- HU 5: La aplicación debe permitir únicamente al usuario con rol de **product owner** registrar, consultar, modificar y eliminar las **historias de usuario** del **product backlog**.

- HU 6: El usuario con rol de **product owner** podrá crear, consultar, modificar y eliminar **Sprints**.
- HU 7: La aplicación debe permitir únicamente al usuario con el rol de **product owner** priorizar las **historias de usuario** en el **product backlog** y **Sprints**.
- HU 8: Los usuarios con roles como **equipo de desarrollo** o **scrum master** únicamente podrán consultar la información de las **historias de usuario** y **Sprints**.
- HU 9: Únicamente el usuario con rol de **product owner** podrá mover las **historias de usuario** en los **Sprints**.
- HU 10: La aplicación no debe permitir que se agreguen **historias de usuario** y **tareas** a un **sprint** con estado en proceso o finalizado.
- HU 11: El usuario con rol de **equipo de desarrollo** podrá asignar una estimación de esfuerzo (tiempo) a cada **historia de usuario**.
- HU 12: Únicamente el usuario con rol de **equipo de desarrollo** podrá dividir las **historias de usuario** en **tareas**.
- HU 13: El usuario con rol de **equipo de desarrollo** podrá consultar, modificar y eliminar las tareas.
- HU 14: La aplicación no debe permitir agregar, modificar o eliminar una **tarea** de la **historia de usuario** cuando el estado del **Sprint** este en **proceso** o **finalizado**.
- HU 15: Únicamente los integrantes del **equipo de desarrollo** podrán asignarse **tareas**.
- HU 16: La aplicación debe permitir crear, consultar, modificar y eliminar **tipos de trabajos** que puedan ser aplicados en las **tareas** por parte del **equipo de desarrollo**.
- HU 17: La aplicación debe permitir únicamente al **equipo de desarrollo** registrar el tiempo invertido en las **tareas** asignadas.

- HU 18: La aplicación debe permitir visualizar un ***burndown chart*** a nivel **proyecto** y **Sprints**.

En los siguientes apartados se mostrarán algunos diagramas los cuales fueron diseñados en base a las historias de usuario presentados anteriormente.

3.1.2 Modelo de negocios

Con este modelo se presentan los procesos de la propia metodología Scrum, esto permite tener una mejor comprensión de los procesos que se llevan a cabo durante la implementación de la metodología Scrum, y a su vez, saber en qué procesos estará involucrada la plataforma a desarrollar (figura 3.1).

En el diagrama de la figura 3.1, se observa cuatro carriles (divisiones verticales) que representan los roles de la metodología Scrum donde cada uno realiza determinados procesos en la metodología, así como los artefactos generados y cambios en la base de datos que se generan en los procesos específicos. Con la identificación de qué procesos son realizados por los roles, permitirá limitar a los usuarios con estos roles, realizar ciertas funciones en la plataforma al momento de gestionar y monitorizar los proyectos de desarrollo de software basados en Scrum.

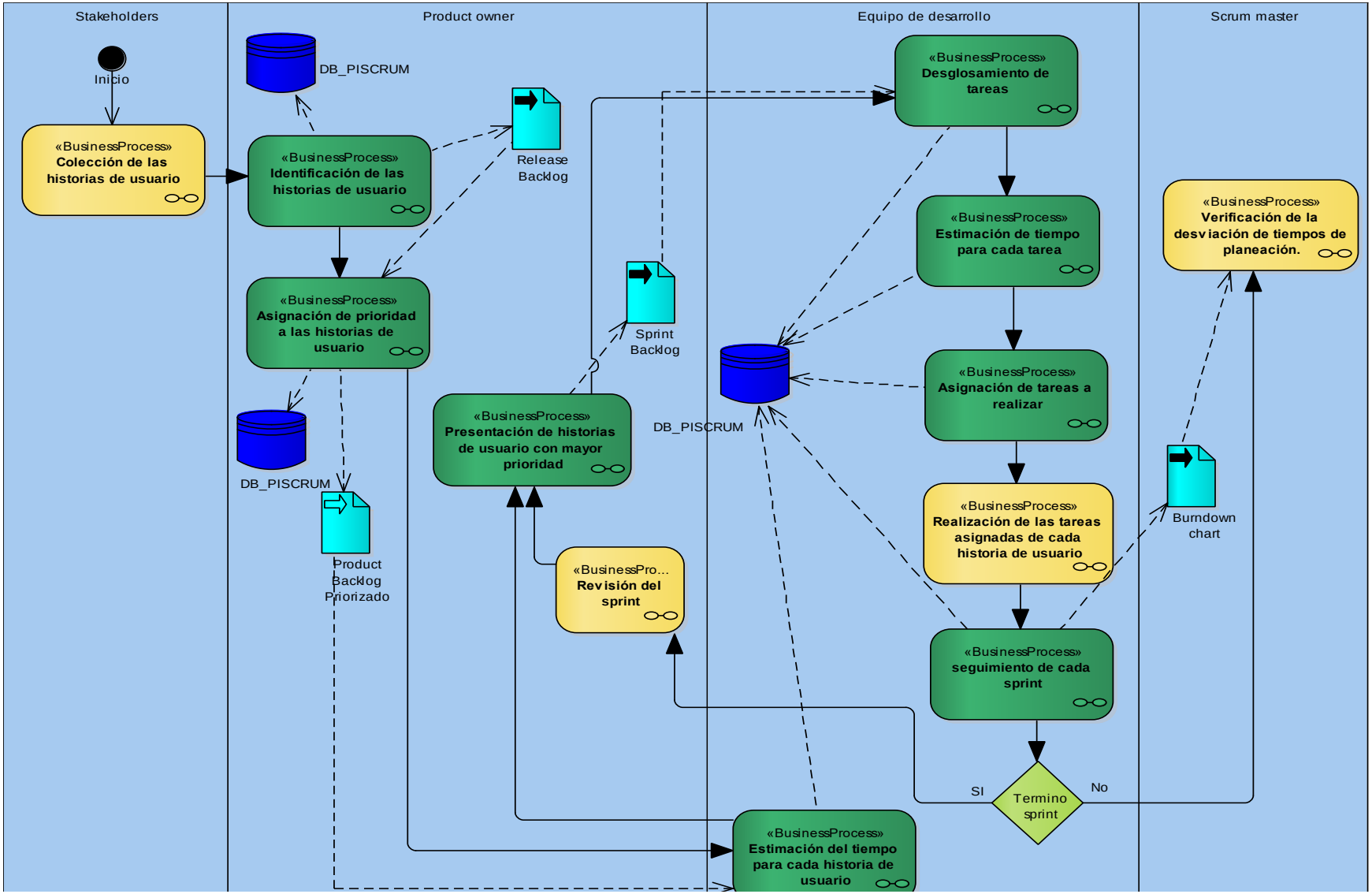


Figura 3.1. Modelado de procesos de negocio de la plataforma.

A continuación, se describe con un nivel de abstracción superior, cada uno de los procesos involucrados del modelo de negocio:

- *Proceso: Colección de las historias de usuario.* El **cliente** (StakeHolder) menciona las **necesidades** o **requisitos (historias de usuario)** al **product owner** y éste realiza la anotación de las mismas.
- *Proceso: Identificación de las historias de usuario.* El **product owner** identifica e interpreta las **historias de usuario** a través de un lenguaje de fácil entendimiento para el **equipo de desarrollo**, además éstas definirán la liberación del producto final.
- *Proceso: Asignación de prioridad a las historias de usuario.* El **product owner** prioriza las **historias de usuario** según al criterio del **cliente** y se obtiene un **product backlog** priorizado.
- *Proceso: Estimación de tiempo para cada historia de usuario.* El **equipo de desarrollo** junto con el **product owner**, estiman los tiempos para el desarrollo de cada **historia de usuario**.
- *Proceso: Presentación de historias de usuarios con mayor prioridad.* Se realiza una reunión antes de iniciar el **Sprint**, en la cual, el **product owner** presenta las **historias de usuarios** con mayor prioridad hasta el momento al **equipo de desarrollo**, dando origen a un **Sprint backlog**.
- *Proceso: Desglosamiento de tareas.* El **equipo de desarrollo** divide en pequeñas **tareas** las **historias de usuarios**.
- *Proceso: Estimación de tiempo para cada tarea.* El **equipo de desarrollo** estima un tiempo para cada **tarea** que surgió del desglosamiento de tareas.

- *Proceso: Asignación de tareas a realizar.* Entre los integrantes del **equipo de desarrollo** se lleva a cabo la asignación de las **tareas** a realizar para completar cada una de las **historias de usuarios**.
- *Proceso: Realización de las tareas asignadas de cada historia de usuario.* El **equipo de desarrollo** dedica tiempo diariamente a cada **tarea** para completar las **historias de usuario** según lo estimado.
- *Proceso: Seguimiento de cada sprint.* Cada integrante del **equipo de desarrollo** registra las horas invertidas en cada **tarea asignada**, esto sirve para saber el avance real de un **proyecto**.
- *Proceso: Revisión del sprint.* El **product owner** comprueba el progreso del **proyecto**, donde identifica las funcionalidades que se pueden considerar hechas y las que no.
- *Proceso: Verificación de la desviación de tiempos de planeación.* El **scrum master** verifica la **desviación de tiempos** de planeación en cada **Sprint** apoyándose con el **burndown chart** y poder mitigar riesgos en caso de existir anomalías en los tiempos estimados, permitiendo que no existan atrasos en las entregas de los **Sprints**.

3.1.3 Modelo de casos de uso

Para el modelado de casos de uso se realizó uno por cada proceso del modelado de negocios en los que interviene la herramienta a desarrollar, la figura 3.2 muestra el proceso de identificación de las historias de usuario descrito con el modelo de casos de uso. A continuación, se describirán los actores que se presentarán a lo largo de los casos de uso a mostrar.

Descripción de los actores de los modelos de casos de uso:

- **Product owner**: Se encarga de obtener las historias de usuario a través de los requerimientos del cliente y presentar las historias de usuario al equipo de desarrollo.

Prioriza las historias de usuario y decide cuales se realizarán en cada uno de los Sprints dependiendo de la prioridad. Además, al finalizar cada de los Sprints revisa si el incremento cumple con lo establecido en un inicio de los Sprints.

- **Equipo de desarrollo**: Son un conjunto de personas encargadas de realizar cada una de las tareas que están involucradas en las historias de usuario y llevando el control del tiempo invertido en las tareas.
- **Scrum master**: Se encarga de monitorizar el flujo que lleva la gestión del proyecto y en caso de existir problemas para cumplir con las historias de usuario, trata de mitigar los problemas para que cada uno de los Sprints se cumpla con el objetivo.

La figura 3.2 ilustra los casos de uso para el **registro de las historias de usuario**.

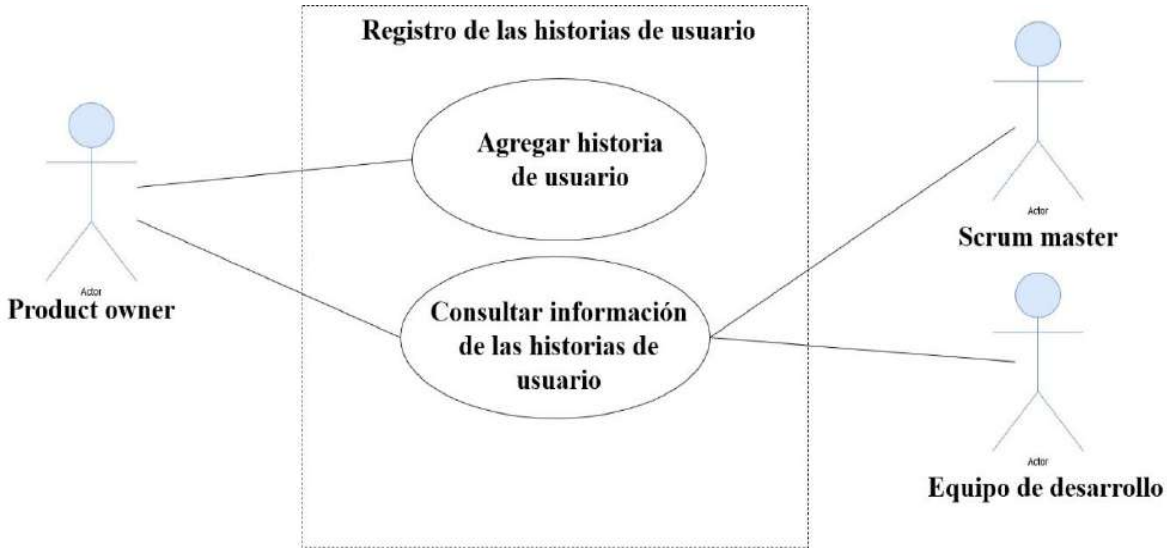


Figura 3.2. Caso de uso para el registro de las historias de usuario.

En la tabla 3.1 se describen los casos de uso ilustrados en la figura 3.2.

Tabla 3.1. Descripción de los casos de uso para el registro de las historias de usuario.

Nombre	Registro de las historias de usuario
Actores	Product owner, Scrum master y Equipo de desarrollo.
Descripción	El **product owner** consulta la información de las **historias de usuario** (**product backlog**). Al consultar la lista, puede verificar, si existe la **historia**

	de usuario que desee agregar al **product backlog**. El **Scrum master** y **Equipo de desarrollo** solo pueden consultar la información de las historias de usuario o en su defecto el **product backlog**.
Datos	Nombre, descripción y un resumen de la historia de usuario.
Respuesta	Confirmación de que la plataforma registro correctamente la historia de usuario.
Comentarios	Para agregar **historias de usuario** al **product backlog** es necesario que el usuario sea autenticado con el rol de **product owner.**

La figura 3.3 muestra los casos de uso para **priorizar las historias de usuario**.

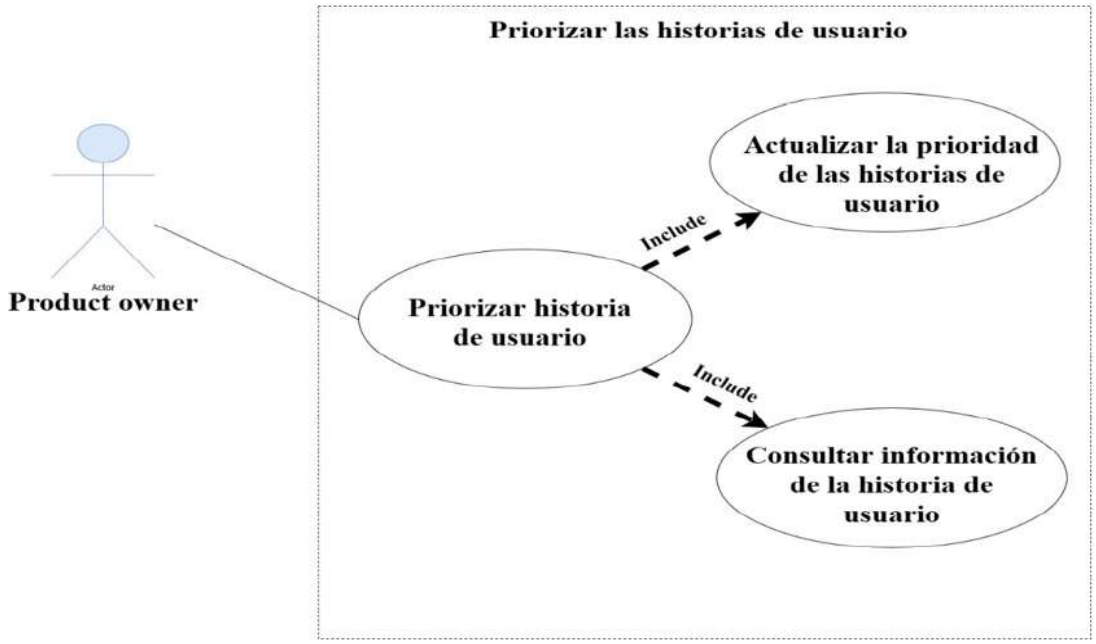


Figura 3.3. Casos de uso para priorizar las historias de usuario.

En la tabla 3.2 se describen los casos de uso ilustrados en la figura 3.3.

Tabla 3.2. Descripción de los casos de uso para priorizar las historias de usuario.

Nombre	Priorizar las historias de usuario
Actores	Product owner.
Descripción	El **product owner** solicita priorizar una **historia de usuario** del **product backlog**, entonces la plataforma consulta información (**orden en la lista**) de la **historia de usuario** a priorizar, y este actualiza la información (**orden en la lista**) de la **historia de usuario** en el **product backlog**.

Datos	El número de orden actual de la historia de usuario.
Respuesta	El product backlog ordenado.
Comentarios	El usuario con rol de **product owner** puede priorizar las historias de usuario en el **product backlog**.

En la figura 3.4 se puede observar los casos de uso para la **presentación del Sprint backlog**.

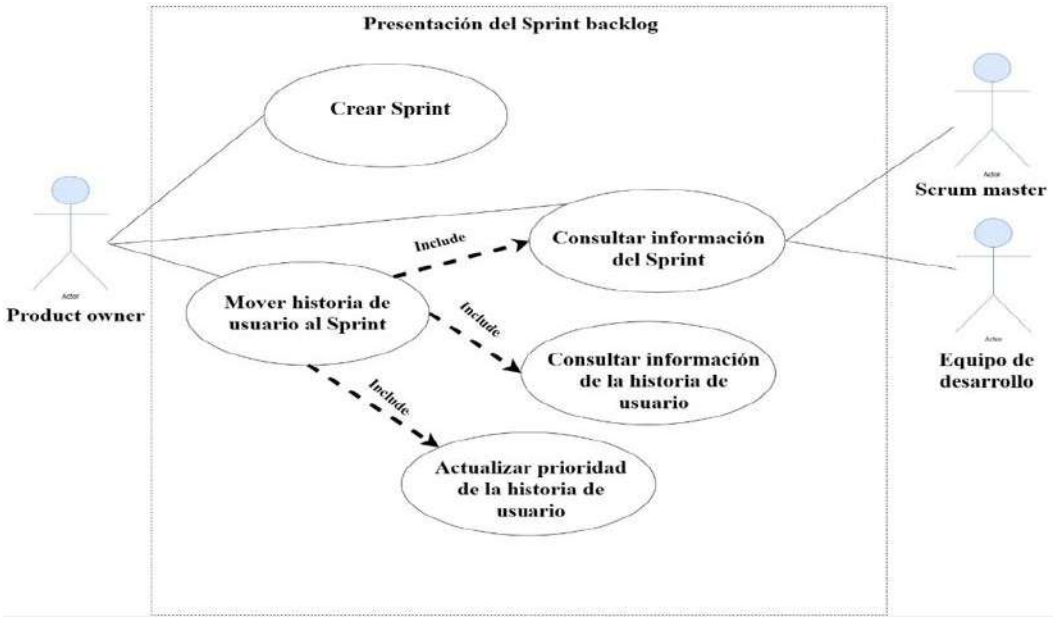


Figura 3.4. Casos de uso para la presentación del Sprint backlog.

En la tabla 3.3 se describen los casos de uso ilustrados en la figura 3.4.

Tabla 3.3. Descripción de los casos de uso para la presentación del Sprint backlog.

Nombre	**Presentación del Sprint backlog**
Actores	Product owner, Scrum master, Equipo de desarrollo.
Descripción	Para presentar las **historias de usuario** que se realizarán en un **Sprint**, el **product owner** solicita la creación de un **Sprint**. Posterior a la creación de **Sprint**, el **product owner** mueve la **historias de usuario** del **product backlog** al **Sprint** creado, para ello la plataforma realiza una consulta del **Sprint** creado y de la **historia de usuario** que será enviada a ese **Sprint**, esto para actualizar la **prioridad de la historia de usuario** que tendrá en el **Sprint**.
Datos	El número de orden actual de la historia de usuario, información del Sprint.

Respuesta	Confirmación de que la plataforma realizo correctamente la creación del Sprint.
Comentarios	El usuario con rol de **product owner** puede crear **Sprints** y mover historias de usuario a los **Sprints**. Los usuarios con rol de **equipo de desarrollo** y **Scrum master** solo pueden consultar información de los **Sprints**.

En la figura 3.5 se puede observar los casos de uso para la **gestión de las tareas**.

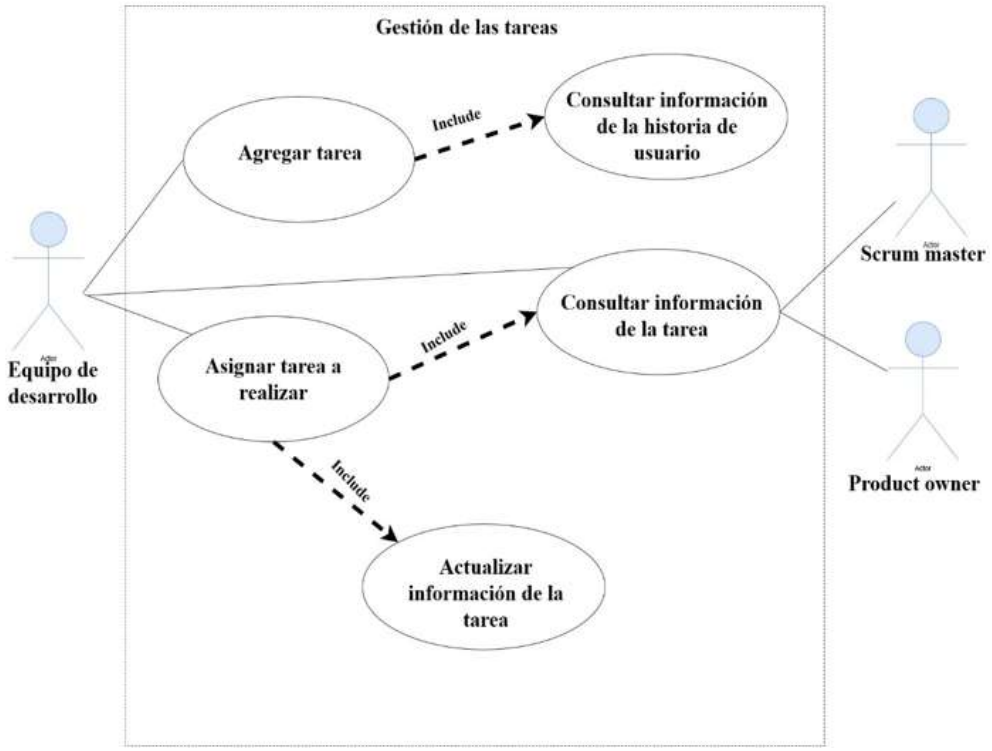


Figura 3.5. Casos de uso para la gestión de las tareas.

En la tabla 3.4 se describen los casos de uso ilustrados en la figura 3.5.

Tabla 3.4. Descripción de los casos de uso para la gestión de las tareas.

Nombre	Gestión de las tareas
Actores	Equipo de desarrollo, Scrum master, Product owner.
Descripción	Un usuario con rol de **equipo de desarrollo** solicita a la plataforma crear una **tarea**, para ello, la plataforma consulta información a que **historia de usuario** será agregada la **tarea**. Los usuarios con rol de **equipo de desarrollo** se asignan las **tareas** a realizar, pero primero la plataforma consulta su información para que sea actualizada (**se asigna el nombre de la persona que realizará la tarea)** de la **tarea**.

Datos	Nombre, descripción, tipo de trabajo a realizar y nombre del integrante del equipo de desarrollo que realizará la tarea.
Respuestas	La plataforma envía un mensaje de éxito cuando se crea una **tarea** o que ha sido asignada.
Comentarios	Los procesos para la gestión de las tareas solamente los puede realizar un usuario con rol de **equipo de desarrollo**, esto se puede apreciar en la figura 3.1 presentada en el modelado de negocios. Los usuarios con rol de **Scrum master** y **product owner** solamente pueden consultar información de las tareas.

En la figura 3.6 se puede observar los casos de uso para la **Estimación de esfuerzo (tiempo) de una tarea.**

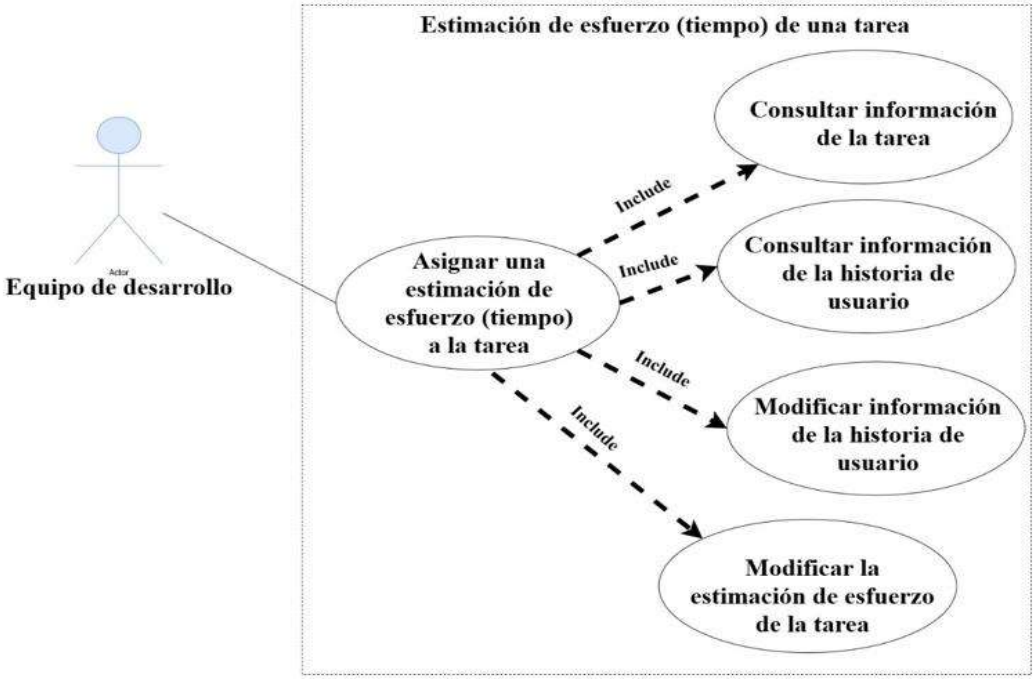


Figura 3.6. Estimación de esfuerzo (tiempo) de una tarea.

En la tabla 3.5 se describen los casos de uso ilustrados en la figura 3.6.

Tabla 3.5. Descripción de los casos de uso para la estimación de esfuerzo de una tarea.

Nombre	Estimación de esfuerzo de una tarea
Actores	Equipo de desarrollo.
Descripción	El usuario con rol de **equipo de desarrollo** consulta la información de la **tarea** para solicitar a la plataforma una asignación de estimación de esfuerzo (tiempo) para la tarea consultada. Cuando se asigna la estimación se modifica la información de la **tarea** y la plataforma consulta información de la **historia de usuario que contiene esta tarea**, que fue estimada, esto para que la estimación de esfuerzo de la **tarea** sea sumada en la **estimación de esfuerzo total de la historia de usuario**.
Datos	Tiempo (horas) de estimación para la tarea.
Respuesta	La plataforma envía un mensaje de éxito indicando que se realizó correctamente la asignación del tiempo a la **tarea**.
Comentarios	Únicamente los usuarios con rol de **equipo de desarrollo** solo pueden asignar una estimación de esfuerzo a la tarea.

3.2 Diseño

En este apartado se presentarán tres diagramas donde el primero muestra la arquitectura lógica de la herramienta propuesta, el segundo exhibe las clases para almacenar la información temporalmente en tiempo de ejecución que la plataforma utiliza y por último se especifica el diagrama de despliegue para mostrar la estructura y conexión de los componentes físicos necesarios para desplegar la herramienta.

3.2.1 Arquitectura de la plataforma

En este apartado se describe detalladamente la arquitectura lógica general de la plataforma a desarrollar. El diagrama de la figura 3.7 muestra los componentes que conforman cada una de las capas de la arquitectura de la plataforma y la manera en que se relacionan.

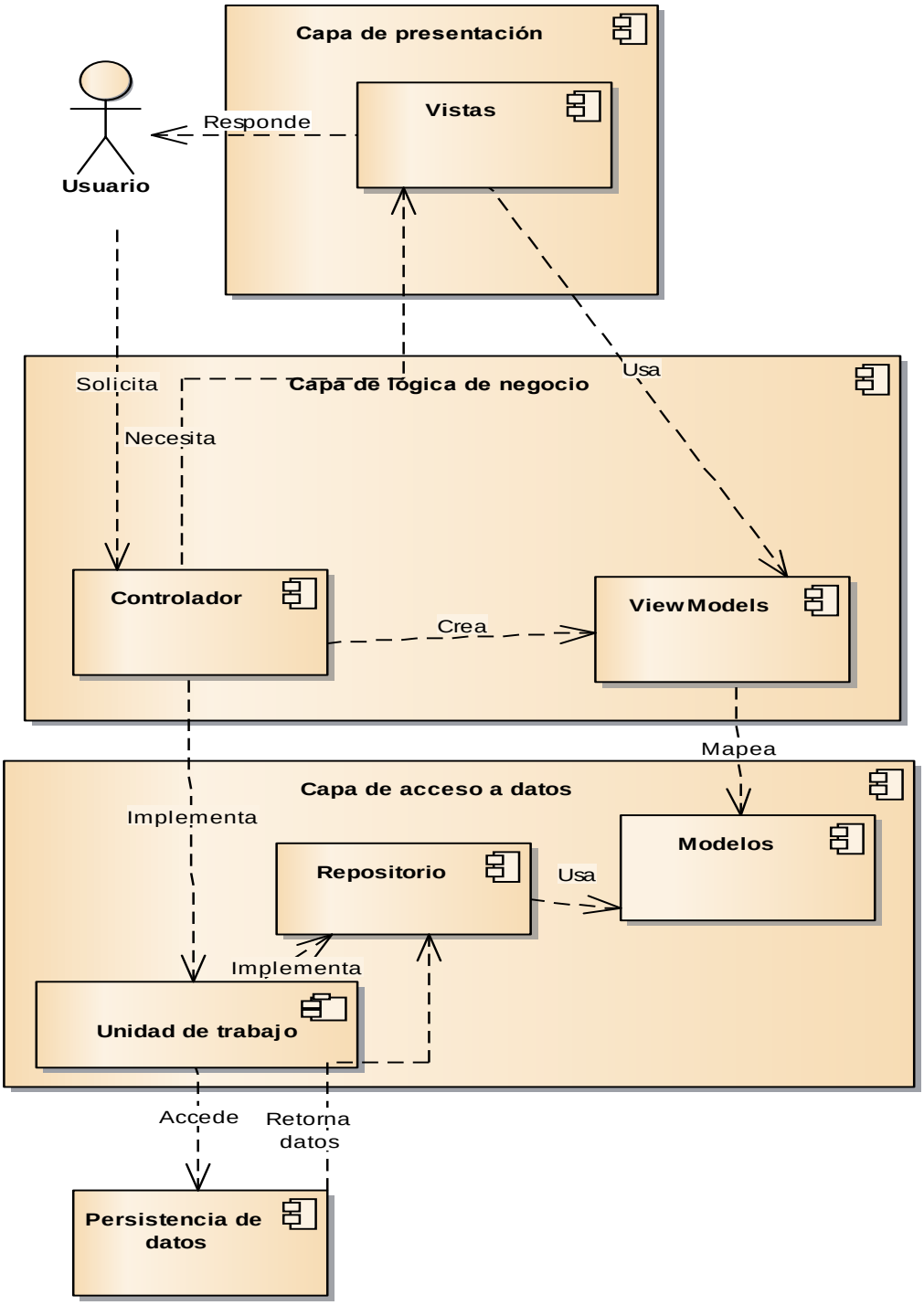


Figura 3.7. Arquitectura de la plataforma implementado el patrón MVC.

Se describe brevemente el flujo que sigue una petición realizada por un usuario a través de los componentes de cada capa de la arquitectura de la herramienta. El **usuario** realiza una petición al **controlador** el cual contiene la lógica de negocio, éste crea un objeto de un **viewmodel** el cual es

una copia exacta de un **modelo** o la combinación de estos para representar un conjunto de datos de diferentes **modelos**, permitiendo crear una capa de abstracción entre los **modelos** que almacenan datos temporalmente y las **vistas**. La construcción de esta capa de abstracción permite proporcionar una mayor seguridad debido a que los **controladores** no manipulan directamente los **modelos**, evitando que se realicen cambios en la **persistencia de datos** no deseados ya que los **modelos** son la representación de las tablas de la **persistencia de datos**. Debido a que el **viewmodel** es creado por el **controlador**, éste puede ser mapeado con el **modelo** o **modelos** correspondientes para que al momento de realizar una transacción utilizando el patrón **unidad de trabajo**, el patrón invoque uno de los métodos (agregar, consultar, modificar y/o eliminar) proporcionados por el patrón de **repositorio**, el cual va a recibir un **modelo** para realizar la transacción con la **persistencia de datos**. En caso de realizar una consulta de datos se lleva a cabo el mismo proceso que cuando un **usuario** efectúa una petición, pero al momento de consultar los datos a la **persistencia de datos**, éste devuelve un conjunto de datos al patrón de **repositorio** que es implementado por el patrón de **unidad de trabajo** que retorna un **modelo** o un conjunto de propiedades de diferentes **modelos** correspondiente a los datos devueltos, en seguida, previo a que llegue el **modelo** o el conjunto de propiedades con datos almacenados temporalmente, son mapeados con un **viewmodel** para que el **controlador** muestre los datos almacenados en el **viewmodel** temporalmente a la **vista**. Como se mencionó en el capítulo 2 la arquitectura a implementar es **MVC**, en la figura 3.7 se observa entre la **capa de presentación** y **lógica de negocio** la relación entre el **controlador**, el **viewmodel** y la **vista** generando la misma estructura y relaciones de **MVC**.

En este apartado del diseño se ha presentado y descrito la arquitectura que contiene la herramienta, explicando la relación entre los componentes de cada capa que compone la arquitectura con el propósito de entender el flujo que sigue una petición a través de estos componentes.

En el siguiente apartado del diseño se presentan las clases que establecen la estructura para almacenar temporalmente la información que es procesada durante el ciclo de vida de la herramienta, así como el modelo de clases de los patrones **unidad de trabajo** y **repositorio**.

3.2.2 Diagrama de clases

Para el modelado de clases se consideraron aquellas clases que participan en los procesos en la etapa de análisis (ver figura 3.8)

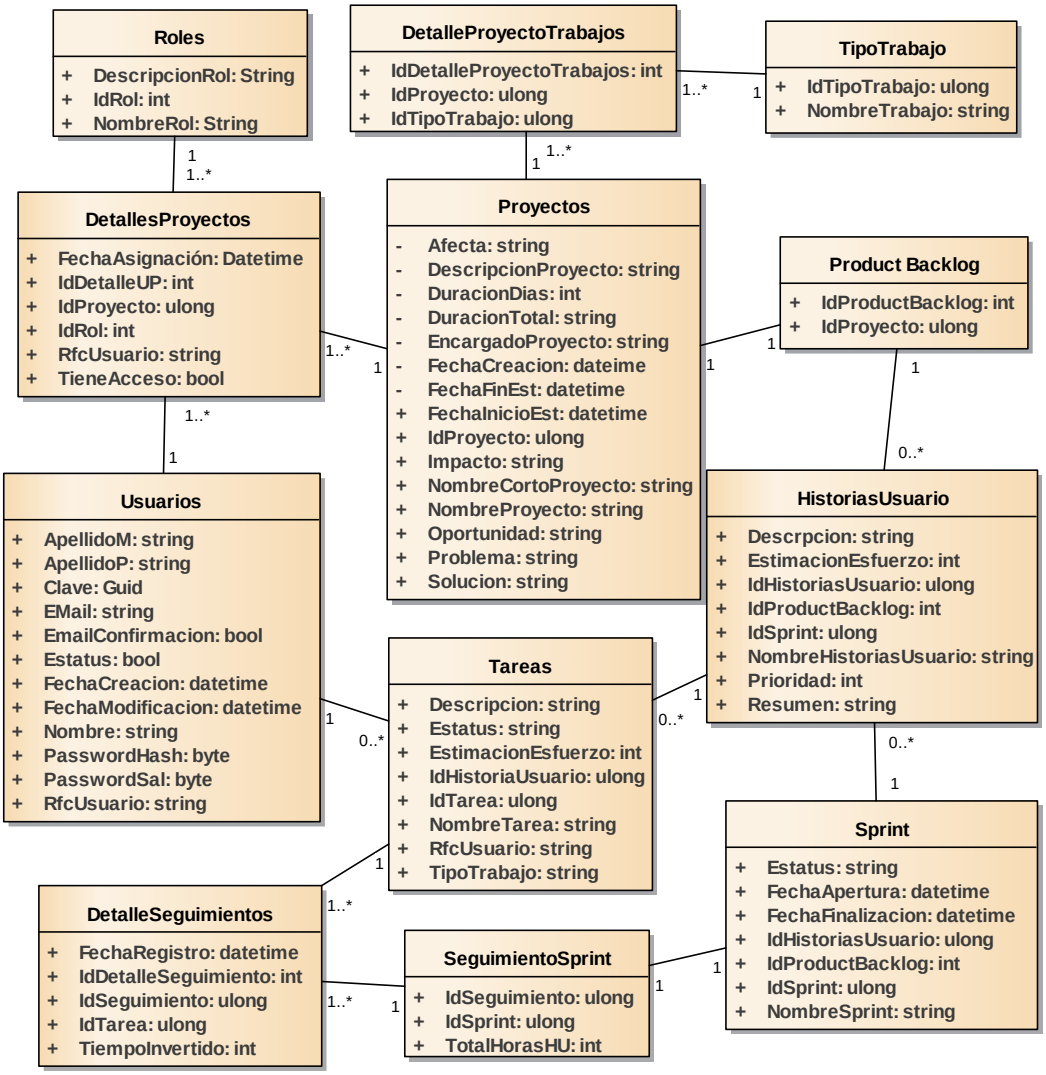


Figura 3.8. Diagrama de clases de la plataforma.

Descripción de las clases del diagrama:

- **Rol**

Esta clase gestiona los datos de los roles que son manipulados para autenticar y autorizar a los usuarios con los roles de la metodología Scrum, permitiendo denegar o dar acceso a diferentes funciones y áreas de la herramienta.

Tabla 3.6. Clase para la gestión de roles.

Atributos	Tipo de dato	Descripción del atributo
IdRol	Entero	Permite almacenar un Id diferente para cada rol y puedan ser identificados como únicos.
NombreRol	Cadena	Guarda el nombre del rol para que este pueda ser identificado en las autenticaciones y autorizaciones dentro del sistema.
DescripcionRol	Cadena	Almacena una descripción del rol para identificar que a detalle cuáles son sus tareas dentro de la aplicación.

- **DetallesProyectos**

Esta clase representa los detalles de los proyectos permitiendo que varios usuarios con diferentes roles puedan estar agregados en un mismo proyecto o diferentes.

Tabla 3.7. Clase para la gestión de los detalles de los proyectos.

Atributos	Tipo de dato	Descripción del atributo
IdDetalleUP	Entero	Almacena un id del detalle para poder ser identificado entre los otros detalles y más rápida la búsqueda de un detalle.
IdProyecto	Entero	Almacena un id de un proyecto para indicar a que proyecto será asignado el usuario

IdRol	Entero	Permite guardar un id de un rol indicando que rol tendrá el usuario en el proyecto a gestionar.
RfcUsuario	Cadena	Guarda el rfc del usuario para indicar que usuario es el asignado al proyecto.
FechaAsignacion	Fecha	Almacena la fecha de asignación al proyecto por parte del product owner.
TieneAcceso	Booleano	Permite almacenar un valor verdadero o falso indicado el estado de acceso al proyecto que fue agregado.

- **Usuarios**

Esta clase representa y almacena los datos de los usuarios para que sean identificados en la herramienta y que tengan acceso a la misma.

Tabla 3.8. Clase para almacenar datos de los usuarios temporalmente.

Atributos	Tipo de dato	Descripción del atributo
RfcUsuario	Cadena	Permite guardar el rfc del usuario para ser identificado dentro de la aplicación
Nombre	Cadena	Guarda el nombre del usuario y este pueda ser identificado por otros usuarios al ser asignado alguna tarea o haber hecho algún movimiento en la gestión de un proyecto.
ApellidoP	Cadena	Guarda el apellido paterno del usuario para poder ser identificados con mayor facilidad por otros usuarios.
ApellidoM	Cadena	Guarda el apellido materno del usuario para poder ser identificados con mayor facilidad por otros usuarios.

Email	Cadena	Almacena un correo electrónico para que el usuario pueda tener acceso a la plataforma.
PasswordHash	Byte	Permite almacenar un Hash de la contraseña clara y con una combinación se pueda obtener una contraseña **Sal** (encriptación).
PasswordSal	Byte	Guarda una contraseña **Sal** para tener mayor protección de la contraseña clara y no sea almacenada tal como se introduce a través de la aplicación.
Clave	Guid	Permite almacenar una clave alfanumérica la cual es generada y actualizada cuando se hace un cambio de contraseña o confirmación de correo electrónico, permitiendo que los enlaces enviados por correo electrónicos sean únicos y puedan ser validados.
Estatus	Booleano	Guarda el estatus del usuario permitiendo dar acceso o no a la aplicación.
EmailConfirmacion	Booleano	Almacena un valor de verdadero o falso para que la aplicación pueda determinar si el correo electrónico introducido por el usuario es real.
FechaCreacion	Fecha	Guarda la fecha y hora de creación del usuario.
FechaModificacion	Fecha	La fecha y hora es almacena en este atributo cuando se realiza una modificación de datos o contraseña.

- **Proyectos**

Esta clase es la representación de los datos de los proyectos para que sean utilizados en la gestión y monitorización de estos.

Tabla 3.9.Clase para la almacenar los datos proyectos gestionados en la plataforma.

Atributos	Tipo de dato	Descripción del atributo
IdProyecto	Entero	Permite almacenar un id para que pueda ser identificado entre los otros registros de proyectos.
NombreProyecto	Cadena	Almacena el nombre del proyecto para que pueda ser identificado por parte de los usuarios involucrados.
NombreCortoProyecto	Cadena	El nombre corto del proyecto es almacenado en este atributo para que pueda ser desplegado en algunas áreas de la aplicación.
DescripcionProyecto	Cadena	Almacena una descripción del proyecto para poder dar un panorama general de este.
FechaInicioEst	Fecha	Permite guardar la fecha de cuando se inició o iniciará el proyecto
FechaFinEst	Fecha	Permite almacenar la fecha de cuando se finalizará el proyecto.
DuracionDias	Entero	Guarda el número de días que tendrá de duración el proyecto en un rango establecido por la fecha de inicio y fin.
DuracionTotal	Cadena	La duración del proyecto en letra es almacenada en este atributo permitiendo identificar con facilidad la duración del proyecto.

EncargadoProyecto	Cadena	Es almacenado el nombre completo del usuario que creo el proyecto.
FechaCreacion	Fecha	La fecha de creación del proyecto es guardada en este atributo.
Problema	Cadena	Permite guardar el problema que genera el desarrollo del proyecto.
Oportunidad	Cadena	Almacena una descripción de oportunidad de negocio que podría tener el proyecto a desarrollar.
Impacto	Cadena	Permite guardar una descripción del impacto que tendrá el proyecto al ser desarrollado.
Afecta	Cadena	Guarda una descripción a quien afecta al ser desarrollado el proyecto.
Solucion	Cadena	Permite almacenar una descripción de la solución del problema a atacar.

- **DetalleProyectoTrabajos**

Esta clase es la representación del detalle de los tipos de trabajos que se pueden realizar en las tareas asignadas al equipo de desarrollo en un proyecto. Esta clase permite que un proyecto pueda tener varios tipos de trabajos.

Tabla 3.10. Clase para la gestión de los detalles de los tipos de trabajos con los proyectos.

Atributos	Tipo de dato	Descripción del atributo
IdDetalleProyectoTrabajos	Entero	Permite almacenar un Id diferente para el detalle de los proyectos y tipos de trabajos relacionados.
IdProyecto	Entero	Guarda un id del proyecto al cual será vinculado un tipo de trabajo.

IdTipoTrabajo	Entero	Un id del tipo de trabajo que es vinculado al proyecto es almacenado en este atributo.

- **TipoTrabajo**

Esta clase corresponde a la representación de los tipos de trabajos que los integrantes del equipo de desarrollo pueden realizar en las tareas asignadas en los proyectos.

Tabla 3.11.Clase para gestionar los diferentes tipos de trabajos para las tareas.

Atributos	Tipo de dato	Descripción del atributo
IdTipoTrabajo	Entero	Permite almacenar un Id diferente para cada tipo de trabajo y puedan ser identificados al relacionarlos con los proyectos.
NombreTrabajo	Cadena	Guarda el nombre del nombre del tipo de trabajo para que este pueda ser identificado al momento de que un integrante del equipo de desarrollo especifique que trabajo realizará en la tarea asignada.

- **Productbacklog**

Esta clase permite almacenar temporalmente datos del product backlog de un proyecto permitiendo gestionar uno con las historias de usuario que no tengan asignado un Sprint.

Tabla 3.12. Clase para gestionar un product backlog.

Atributos	Tipo de dato	Descripción del atributo
IdProductBacklog	Entero	Permite almacenar un id para el product backlog de cada proyecto esto para poder determinar que historias de usuarios pertenecen algún proyecto.
IdProyecto	Entero	Guarda el id del proyecto para poder vincularlo con un product backlog.

- **HistoriasUsuario**

Esta clase representa los datos de las historias de usuario que hacen el producto, las cuales sirven para ser procesadas durante las etapas implicadas en el gestión y monitorización de proyectos basado en la metodología Scrum.

Tabla 3.13. Clase para la gestión de las historias de usuario.

Atributos	Tipo de dato	Descripción del atributo
IdHistoriaUsuario	Entero	Permite almacenar un Id para identificar cada una de las historias de usuario en el proyecto y los demás proyectos.
NombreHistoriaUsuario	Cadena	Guarda el nombre de la historia de usuario para poder visualizarlo en el product backlog.
DescripcionHistoriaUsuario	Cadena	Almacena una descripción para proporcionar un panorama general de la historia de usuario y lo puedan entender los integrantes del equipo de desarrollo.
ResumenHistoriaUsuario	Cadena	Permite almacenar un resumen para proporcionar un mayor detalle de las historias de usuario para que tener un mejor entendimiento.
EstimacionEsfuerzo	Entero	Las horas de esfuerzo requerido para poder realizar la historia de usuario son almacenadas en este atributo.

Prioridad	Entero	Permite guardar un numero de prioridad en el product backlog o Sprints.
IdProductBacklog	Entero	Almacena el id del product backlog al cual pertenece la historia de usuario.
IdSprint	Entero	Guarda el id del Sprint al cual pertenece la historia de usuario cuando es presentada en el Sprint backlog.

- **Sprint**

Esta clase es la representación de los datos de los Sprints, permitiendo crearlos y se puedan presentar las historias de usuario con mayor prioridad que serán desarrolladas durante el Sprint creado.

Tabla 3.14. Clase para gestionar Sprints en los proyectos.

Atributos	Tipo de dato	Descripción del atributo
IdSprint	Entero	Permite almacenar un Id para identificar los Sprints de los proyectos.
NombreSprint	Cadena	Guarda el nombre del Sprint para que pueda ser identificado dependiendo del módulo a desarrollar.
FechaApertura	Fecha	Almacena una fecha indicando cuando inicio el Sprint.
FechaFinalizacion	Fecha	Permite almacenar una fecha indicando cual va a finalizar o finalizó el Sprint.

Estatus	Cadena	El estatus del Sprint es almacenado en este atributo, permitiendo identificar si ha iniciado, finalizado ó está inactivo el Sprint.

- **Tareas**

Esta clase representa los datos de las historias de usuario que hacen el producto, las cuales sirven para ser procesadas por las etapas implicadas en el gestión y monitorización de proyectos basado en la metodología Scrum.

Tabla 3.15. Clase para gestionar tareas en las historias de usuario.

Atributos	Tipo de dato	Descripción del atributo
IdTarea	Entero	Permite almacenar un Id para identificar cada una de las tareas en el proyecto y los demás proyectos.
NombreTarea	Cadena	Guarda el nombre de la tarea para poder visualizarlo en las historias de usuario.
Descripcion	Cadena	Almacena una descripción para proporcionar un panorama general de la tarea y lo puedan entender los integrantes del equipo de desarrollo.
TipoTrabajo	Cadena	Permite almacenar el tipo de trabajo que realizará la persona asiganada en la tarea.
EstimacionEsfuerzo	Entero	Las horas de esfuerzo requerido para poder realizar la tarea
Prioridad	Entero	Permite guardar un numero de prioridad en las historias de usuario.

Estatus	Cadena	Almacena un estatus para poder identificar que tareas han sido realizadas, cuando están pendientes o en proceso.
IdHistoriaUsuario	Entero	Guarda el id de la historia de usuario permitiendo identificar que tareas pertenecen a la historia de usuario.
RfcUsuario	Cadena	Permite almacenar el rfc del usuario que realizará la tarea durante el desarrollo.

- **SeguimientoSprint**

Esta clase representa los datos de un seguimiento del Sprint permitiendo poder realizar varios seguimientos dependiendo del número de los Sprints en un proyecto.

Tabla 3.16. Clase para realizar el seguimiento del Sprint.

Atributos	Tipo de dato	Descripción del atributo
IdSeguimiento	Entero	Guarda un id para poder identificar el seguimiento y se puedan extraer los datos del seguimiento para generar el **burndown chart**.
IdSprint	Entero	Guarda un id del Sprint para vincularlo con un seguimiento y poder determinar a qué Sprint pertenece el seguimiento.
TotalHorasHU	Entero	Almacena el total de horas requeridas para realizar las historias de usuario que están dentro del Sprint.

- **DetalleSeguimiento**

Se representa el detalle del seguimiento del Sprint el cual va a permitir registrar las horas invertidas en las tareas asignas a los integrantes del equipo de desarrollo, permitiendo generar el **burndown chart**.

Tabla 3.17. Clase para gestionar el registro del tiempo invertido en las tareas.

Atributos	Tipo de dato	Descripción del atributo
IdDetalleSeguimiento	Entero	Permite almacenar un Id para cada detalle de un seguimiento para ser consultados con mayor rapidez.
IdSeguimiento	Entero	Guarda un id del seguimiento el cual está vinculado con un Sprint para poder determinar a qué Sprint pertenece el seguimiento.
IdTarea	Entero	Este id almacenado representa la tarea a la cual se le registrarán el tiempo invertido.
TiempoInvertido	Entero	Los integrantes del equipo de desarrollo registran el tiempo invertido en las tareas asignadas.
FechaRegistro	Fecha	La fecha de registro del tiempo invertido por parte de los integrantes del equipo de desarrollo es almacenada en este atributo.

Las clases que se describieron anteriormente proporcionan una estructura para almacenar la información temporalmente durante el ciclo de vida de la plataforma. En la figura 3.8 se puede observar que las clases no cuentan con métodos, esto es debido a que para realizar operaciones con

ellas se implementan los patrones unidad de trabajo y repositorio. A continuación, se describirá la función de los patrones.

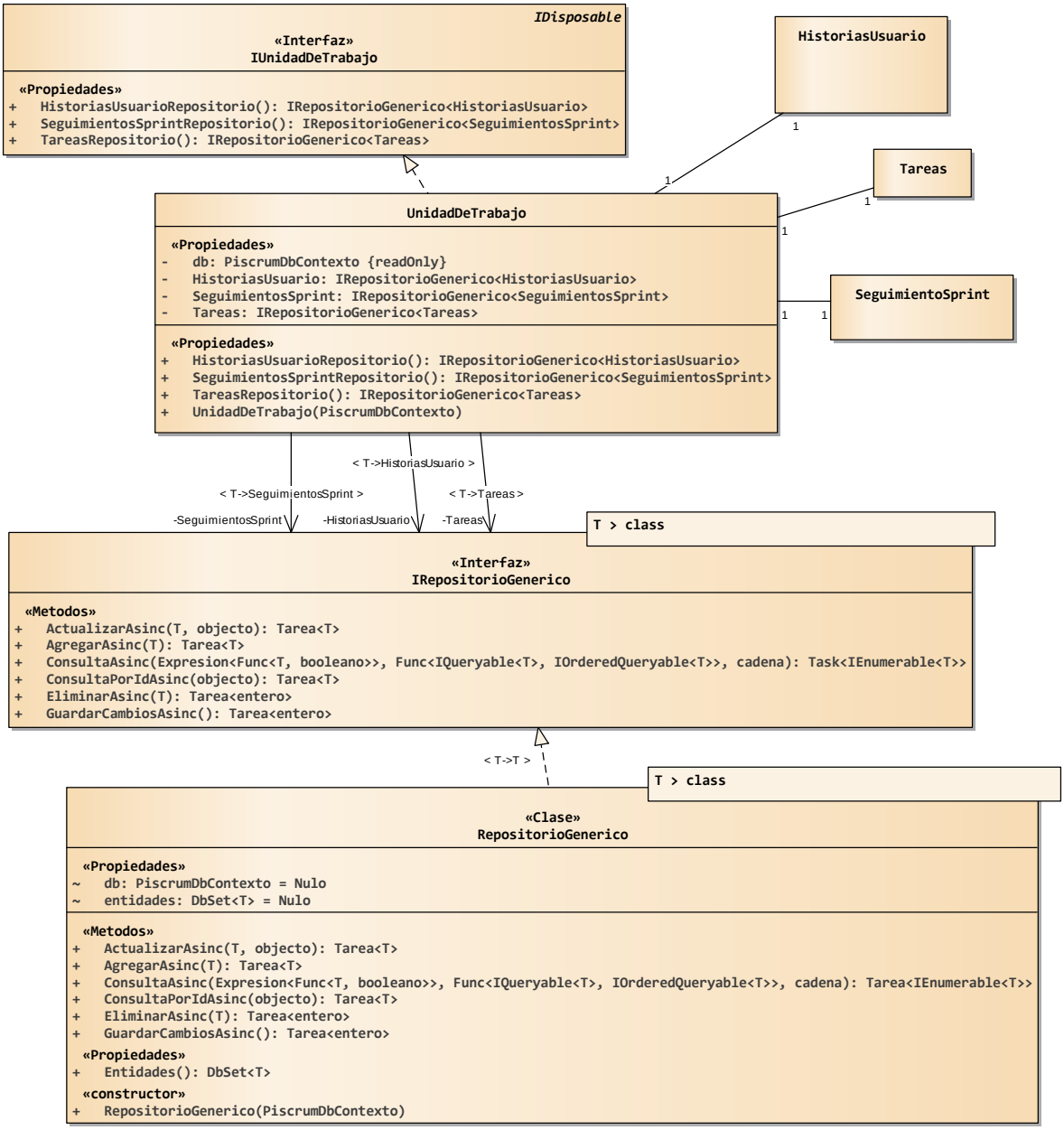


Figura 3.9. Diagrama de clases de los patrones UnidadDeTrabajo y Repositorio.

En la figura 3.9 se muestran las clases y atributos que representan el seguimiento del Sprint. Como se observa en la figura 3.9, se modelan dos clases y dos interfaces que representan los patrones de diseño **RepositorioGenerico** y **UnidadDeTrabajo,** donde la primera tiene la función

de crear una capa de abstracción entre la capa de acceso a datos y la lógica de negocio. Esta capa de abstracción implementa las operaciones CRUD (Create, Read, Update, Delete) de manera asíncrona, permitiendo hacer peticiones a la base de datos disminuyendo el tiempo de respuesta, esta capa hereda de la interfaz **IRepositorioGenerico,** la cual implementa los mismos métodos de **RepositorioGenerico,** pero esto permite la abstracción del mecanismo de los métodos que implementa **RepositorioGenerico**. **UnidadDeTrabajo** permitirá manejar transacciones durante la manipulación de datos utilizando los métodos implementados en el patrón **RepositorioGenerico**. Al realizar operaciones en la base de datos **UnidadDeTrabajo,** se mantiene una lista de los objetos afectados (clases) por una transacción de negocio, coordinando la escritura de los cambios y la resolución de problemas de concurrencia.

3.2.3 Diagramas de despliegue

El diagrama de despliegue permite visualizar la estructura (componentes físicos y software) necesaria para que la aplicación a desarrollar pueda ser organizada al momento de ser implantada (ver figura 3.10).

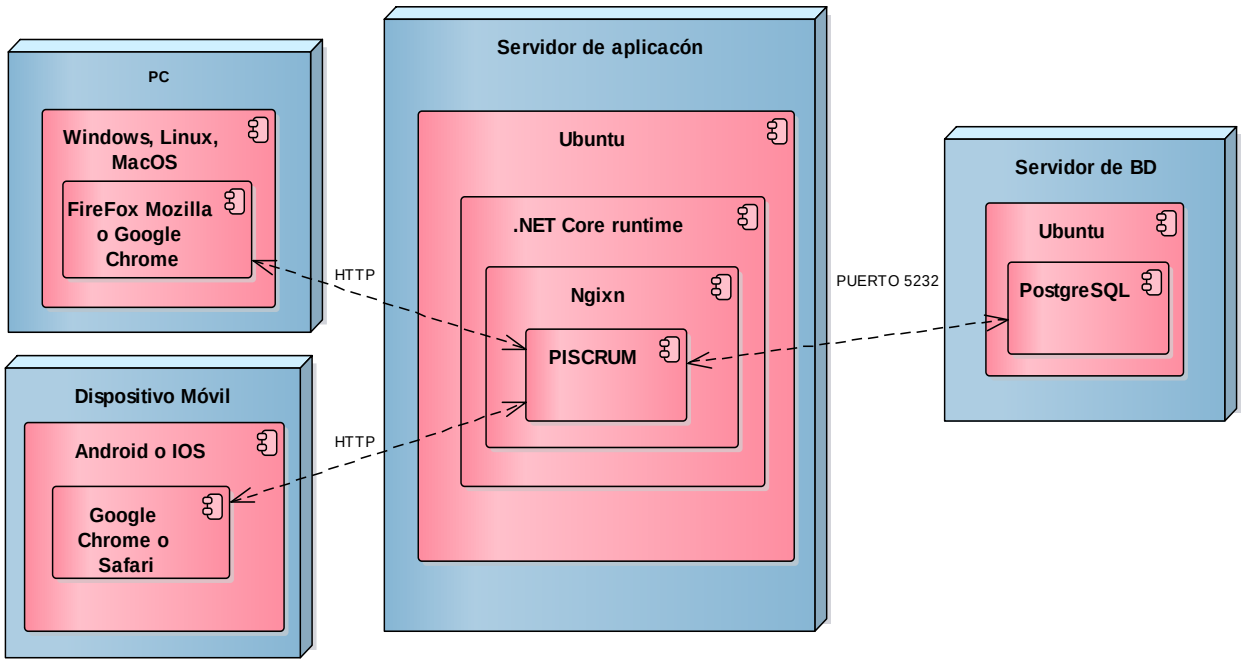


Figura 3.10. Diagrama de despliegue de la plataforma.

- **PC:** Es el dispositivo en el cual se manipulará la plataforma con nombre **PISCRUM** por medio del **navegador Web** (FireFox Mozilla o Google Chrome), el cual se tiene que instalar en la **PC** bajo cualquiera de las siguientes plataformas operativas: Windows 8.1, 10, Linux ó MacOS.
- **Dispositivo Móvil:** Es el dispositivo en el cual se manipulará la plataforma **PISCRUM** por medio del **navegador Web** (FireFox Mozilla o Google Chrome), el cual se tiene que instalar en el dispositivo móvil y que puede contener los siguientes sistemas operativos: Android o IOS.
- **Servidor de aplicación:** El servidor de aplicación contiene el sistema operativo Linux en su distribución **Ubuntu 18.04**, en el cual se tiene instalado el **.NET Core runtime 2.2** para ejecutar aplicaciones **.Net**. En el servidor de aplicación estará desplegada la plataforma **PISCRUM,** la cual necesita el servidor Web **Ngixn**. **PISCRUM** podrá ser consultada por los dispositivos mencionados anteriormente a través del **navegador Web**.
- **Servidor de base de datos:** En este servidor se tendrá instalado el manejador de base de datos: **postgreSQL 9.5**, el cual es el encargado de administrar y manipular los datos. En dicho servidor se tiene instalado el sistema operativo Linux en su distribución **Ubuntu**.

En la figura 3.11 se ilustra la estructura (componentes físicos y software) necesaria para que la aplicación a desarrollar pueda ser organizada al momento de ser desarrollada.

- **PC de desarrollo:** La PC de desarrollo contiene el sistema operativo **Windows 10**, en el cual se tiene instalado el .**NET Core SDK 2.2** para ejecutar aplicaciones .NET. En la PC de desarrollo se tiene instalado **Servicios de Información de Internet** (en inglés **Internet Information Services** o **IIS**) versión 10.0, convirtiendo la PC en un servidor

web ofreciendo los servicios de HTTP, SMTP y HTTPS, permitiendo que la plataforma PISCRUM pueda ser consultada a través del navegador web **FireFox Mozilla**. Para la administración y manipulación de los datos, se tiene instalado el gestor de base de datos PostgreSQL 9.5.

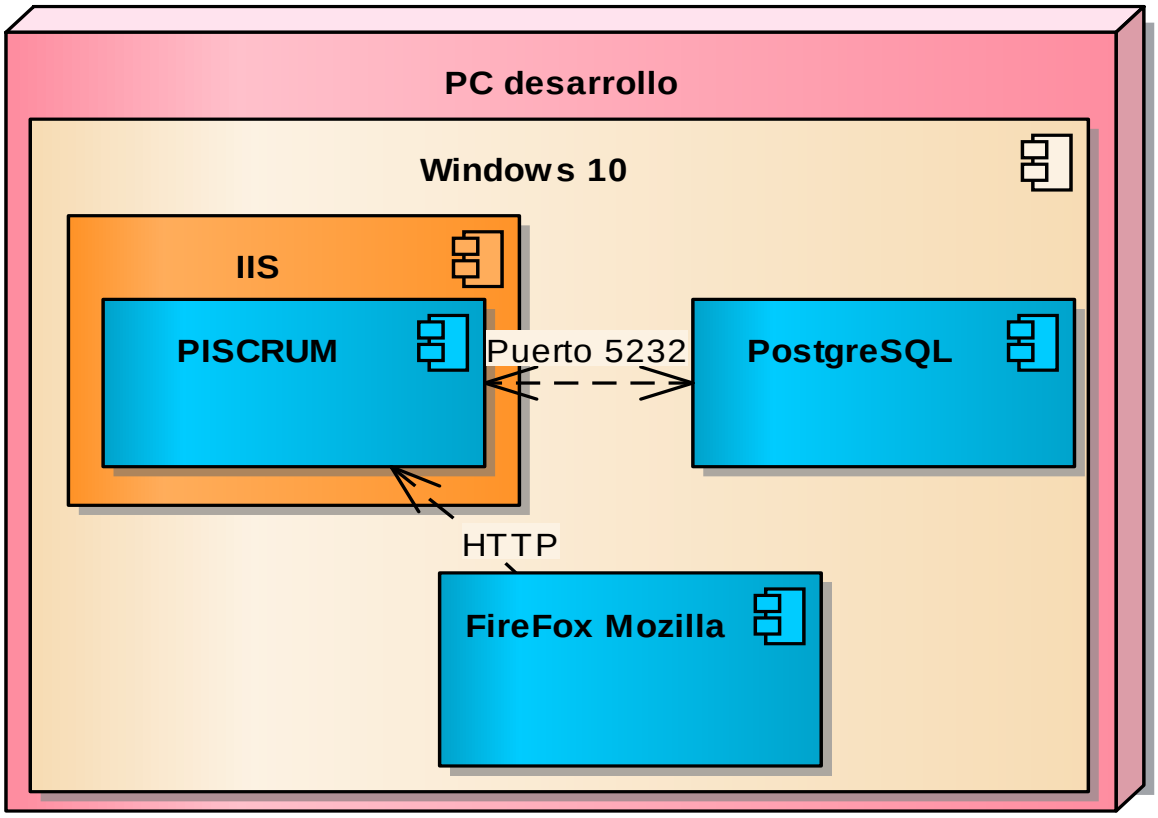


Figura 3.11. Diagrama de despliegue de la plataforma en la PC de desarrollo.

En este capítulo se ha estudiado la arquitectura general de la plataforma y sus diferentes procesos con el objetivo de identificar en que fases de la metodología Scrum (ver figura 3.1 que se mostró en la sección **Modelado de negocios**) son en las que participa la plataforma propuesta en este trabajo de tesis, además se han presentado los modelos diseñados para el desarrollo de la misma. En el siguiente capítulo se expondrá el proceso de desarrollo del trabajo propuesto en esta tesis haciendo uso de diversas tecnologías descritas en el capítulo 2 y siendo soportado por el análisis y diseño realizado en este capítulo.

Capítulo 4 Implementación

El desarrollo de esta plataforma será en 5 Sprints (ver tabla 4.1), donde en el primer Sprint se presenta la implementación de la **capa de acceso a datos**, así como la implementación de los patrones **unidad de trabajo** y **repositorio**. En el segundo Sprint se implementa la **autenticación y autorización de los usuarios** que accederán a la plataforma y la **creación de proyectos**, en el tercer Sprint se presenta la **gestión de las historias de usuarios** y **Sprints** de los proyectos. Para el cuarto Sprint se muestra la implementación de la **división de las historias de usuario en tareas** y su **asignación al equipo de desarrollo**, finalmente en el quinto Sprint se implementa el **seguimiento del sprint** y la **graficación** de este.

Al inicio de cada Sprint el **product owner**, el **equipo de desarrollo** y el **Scrum master** se reunieron para realizar la **planeación del Sprint**, donde el **product owner** presentó cada **Sprint backlog** (conjuntos de historias de usuario priorizadas). Posteriormente a la presentación del **Sprint Backlog**, el **equipo de desarrollo** y el **Scrum master** establecieron la duración de cada **Sprint** durante estas reuniones. Las planeaciones que se realizaron dio como resultado la tabla 4.1. Cabe mencionar que las **historias usuario** contenidas en cada **Sprint backlog**, fueron presentadas en el capítulo 3 y priorizadas por el **product owner**. El tiempo que tomará cada una de las historias de usuario y sus tareas se detallará en cada Sprint.

Tabla 4.1. Planeación de los Sprints de la plataforma.

Sprint	Historias de usuario	Fecha de inicio	Fecha de finalización	Duración del Sprint (semanas)
1		01-05-2019	30-06-2019	8
2	HU1, HU2, HU3 Y HU4	01-07-2019	03-08-2019	5
3	HU5, HU6, HU7, HU8, HU9 Y HU10	04-08-2019	19-10-2019	11

4	HU11, HU12, HU13, HU14, HU15 Y HU16	20-10-2019	11-11-2019	3
5	HU17 Y HU18	12-11-2019	26-11-2020	2
Duración total del desarrollo de la plataforma				7 meses y una semana

Se puede observar que el **Sprint 1** de la tabla 4.1, no contiene **historias de usuario** esto es debido a que como se mencionó anteriormente, es la creación de la **capa de acceso a datos** y la implementación de los patrones **unidad de trabajo** y **repositorio**, lo cual no está dentro de las **historias de usuario**. Pero es necesario su desarrollo y presentación porque es parte de la base para la plataforma propuesta.

Antes de iniciar la descripción de cada Sprint es necesario presentar la configuración necesaria para el desarrollo de plataforma, así como para su implantación.

4.1 Configuración requerida para el desarrollo e implementación de la plataforma

En las tablas 4.2 y 4.3 se describen los requerimientos de hardware, programas y utilidades necesarias para el desarrollo e implantación de la plataforma propuesta.

Tabla 4.2. Requerimientos de hardware, programas y utilidades para el desarrollo de la plataforma.

PC DE DESARROLLO			
Software			
Programa	**Versión**	**Sitio de descarga**	**Año de descarga**
Visual Studio	Community 2019	https://visualstudio.microsoft.com/es/downloads/?rr=https%3A%2F%2Fwww.google.com%2F	2019
PostgreSQL	9.5	https://www.enterprisedb.com/downloads/postgres-postgresql-downloads	2019

PgAdmin	3	https://www.pgadmin.org/download/pgadmin-3-windows/	2019
Fire Fox Mozilla	75.0	https://www.mozilla.org/en-US/firefox/new/	2019
Características de la PC			
S.O	Procesador	Memoria RAM	Disco Duro
Windows 10 de 64 bits	Intel Core i7 a 2.40 GHz quinta generación	16 GB	1 TB de estado solido

Tabla 4.3. Requerimientos de hardware, programas y utilidades para el despliegue de la plataforma.

SERVIDOR DE APLICACIÓN			
Software			
Programa	**Versión**	**Sitio de descarga**	**Año de descarga**
.NET Core runtime	2.2	https://docs.microsoft.com/es-es/dotnet/core/install/linux-package-manager-ubuntu-1910	20019
Ngixn	18.04	http://nginx.org/en/linux_packages.html#Ubuntu	2019
PostgreSQL	9.5	https://www.enterprisedb.com/downloads/postgres-postgresql-downloads	2019
Características del servidor de aplicación			
S.O	Procesador	Memoria RAM	Disco Duro
Linux Ubuntu 18.04	Intel Xeon inside decima octava generación	8 GB	120 GB

Antes de iniciar con los Sprints es necesario la instalación del SDK para la creación de aplicaciones web bajo el marco de desarrollo ASP.NET Core MVC.

4.2 Instalación del .NET Core SDK 2.2

Como se especificó en el capítulo 2 el IDE utilizado para la implementación de la plataforma es Visual Studio, por lo cual es necesario tener instalada la plantilla de ASP.NET Core 2.2 para poder de desarrollar aplicaciones web. El primer paso para obtener la plantilla es necesario acceder a la siguiente página oficial de Microsoft **https://dotnet.microsoft.com/download/dotnet-core/2.2** (ver figura 4.1).

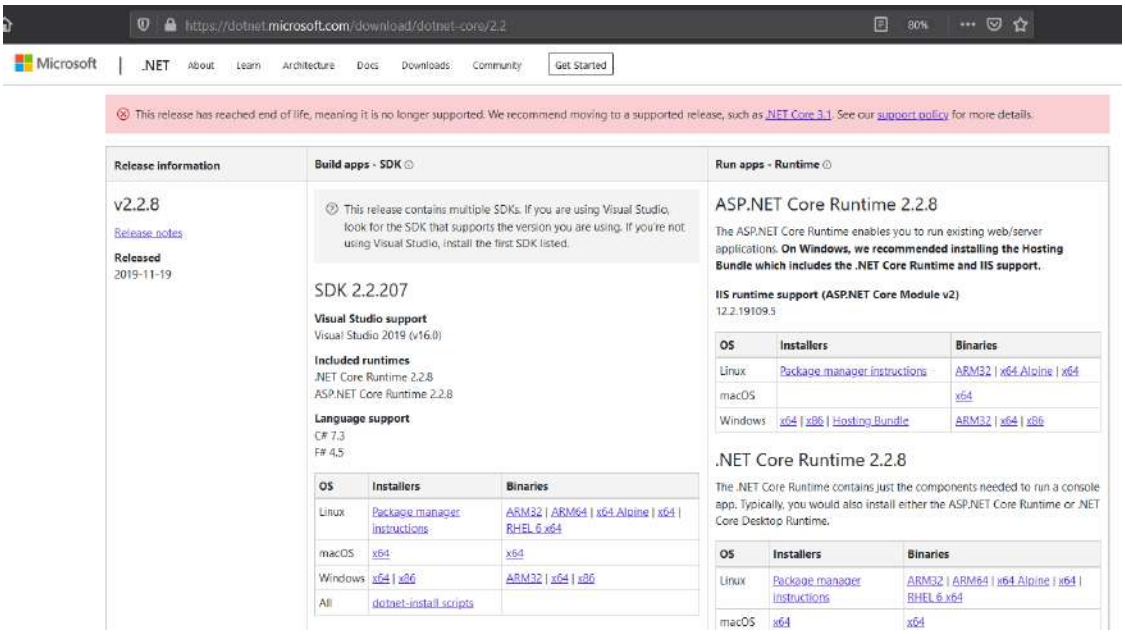


Figura 4.1. Página oficial para descargar el SDK de ASP.NET Core (2019).

En la figura 4.1 se observan tres secciones para descargar determinado paquete, para el desarrollo de aplicaciones web con .NET Core es necesario instalar el SDK. Cuando la plataforma propuesta sea implantada solamente se instalará el runtime de .NET Core en el servidor de aplicación. Cada uno de estos paquetes pueden ser instalados en diferentes plataformas (Windows, MacOS o Linux) para diferentes arquitecturas, para el desarrollo del proyecto para el sistema

operativo Windows, la instalación del SDK es sencilla, simplemente seleccionando el botón **install** (β) (ver figura 4.2).

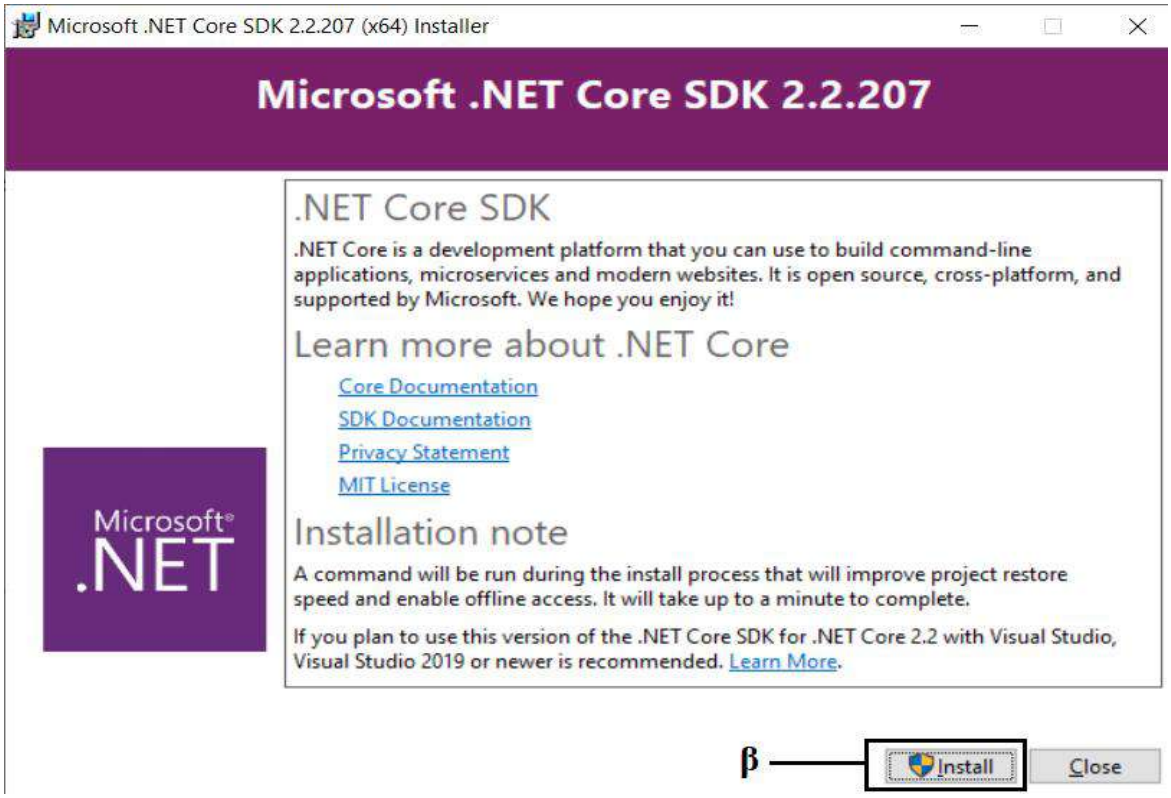


Figura 4.2.Ventana del asistente de instalación.

Figura 4.3. Ventana del asistente de instalación donde se muestra el progreso de instalación.

Después de haber seleccionado el botón **install** se muestra una ventana donde se observará una barra de progreso de instalación (ver figura 4.3), después de haber realizado la instalación es recomendable reiniciar el sistema operativo del pc de desarrollo para que se puedan hacer correctamente los cambios en Visual Studio.

A continuación, se describirá el proceso de la creación de la instancia (solución) en Visual Studio que contendrá el proyecto para el desarrollo de la plataforma. Cuando se ejecuta la aplicación Visual Studio proporciona diferentes opciones para trabajar en un proyecto (ver figura 4.4), para el desarrollo de la plataforma se creó un nuevo proyecto para ello se seleccionó la opción de **Crear un proyecto** (β).

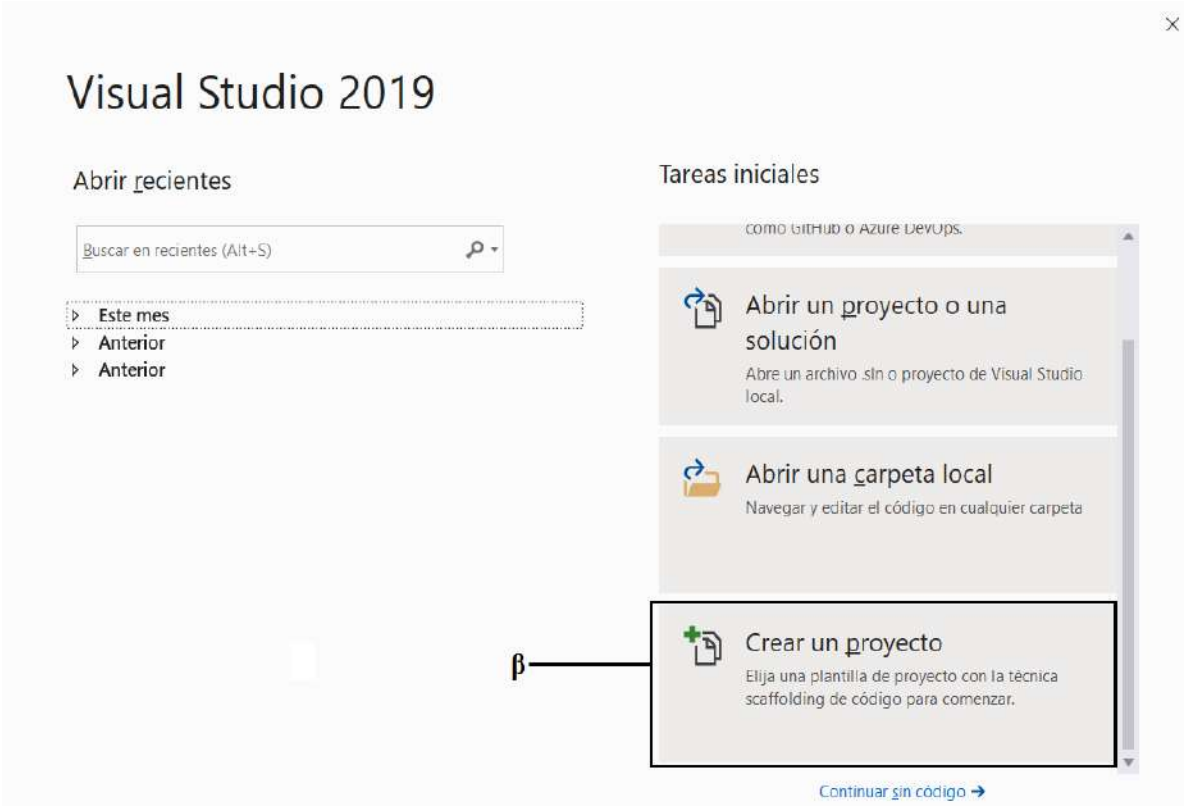


Figura 4.4. Asistente de Visual Studio 2019 para trabajar en un proyecto.

Después de haber seleccionado la opción de crear un proyecto, el asistente muestra una ventana donde permite crear diferentes tipos de proyectos dependiendo la necesidad, para el desarrollo de la plataforma se usará **aplicación web ASP.NET Core** (β) como se observa en la figura 4.5.

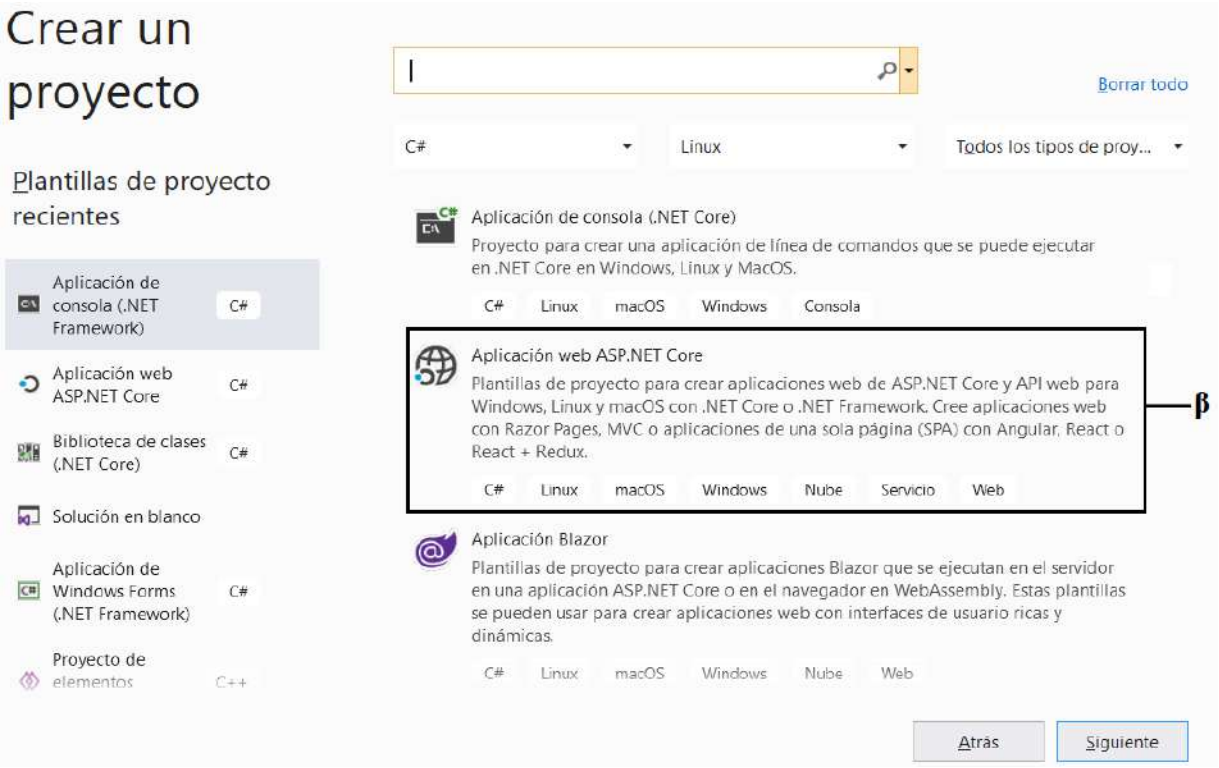


Figura 4.5. Ventana para seleccionar el tipo de proyecto con Visual Studio.

Posterior haber seleccionado el tipo de proyecto es necesario dar un nombre al proyecto (ver indicador 1) y por último seleccionar el botón **Crear** (ver indicador 2) como se ilustra en la figura 4.6. La solución toma el mismo nombre del proyecto.

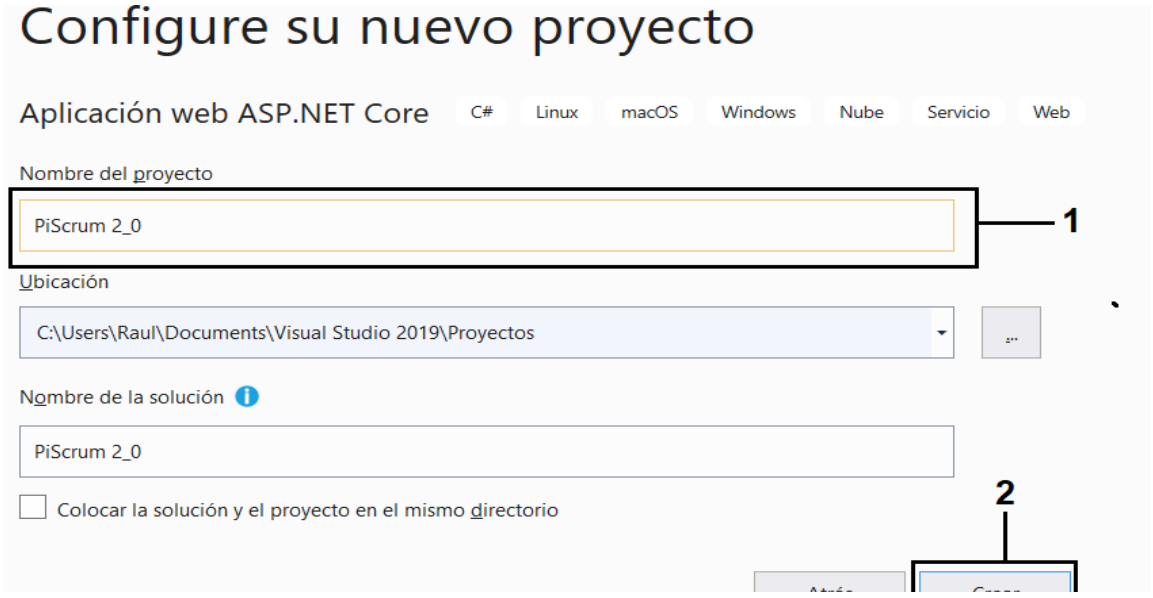


Figura 4.6. Ventana para proporcionar un nombre al proyecto.

Por último, después de haber seleccionado el botón **Crear** (ver figura 4.6), el asistente de Visual Studio muestra una ventana donde se puede trabajar con diferentes tipos de plantillas en una aplicación web ASP.NET Core, para el desarrollo de la plataforma se usará la plantilla **Aplicación web (controlador de vista de modelos)** (ver indicador 1) esta plantilla proporciona un proyecto pre configurado con un controlador y sus vistas (ver figura 4.7). Con esto al seleccionar el botón **Crear** (ver indicador 2), Visual Studio preparará el entorno de trabajo para el desarrollo de la plataforma en la instancia creada.

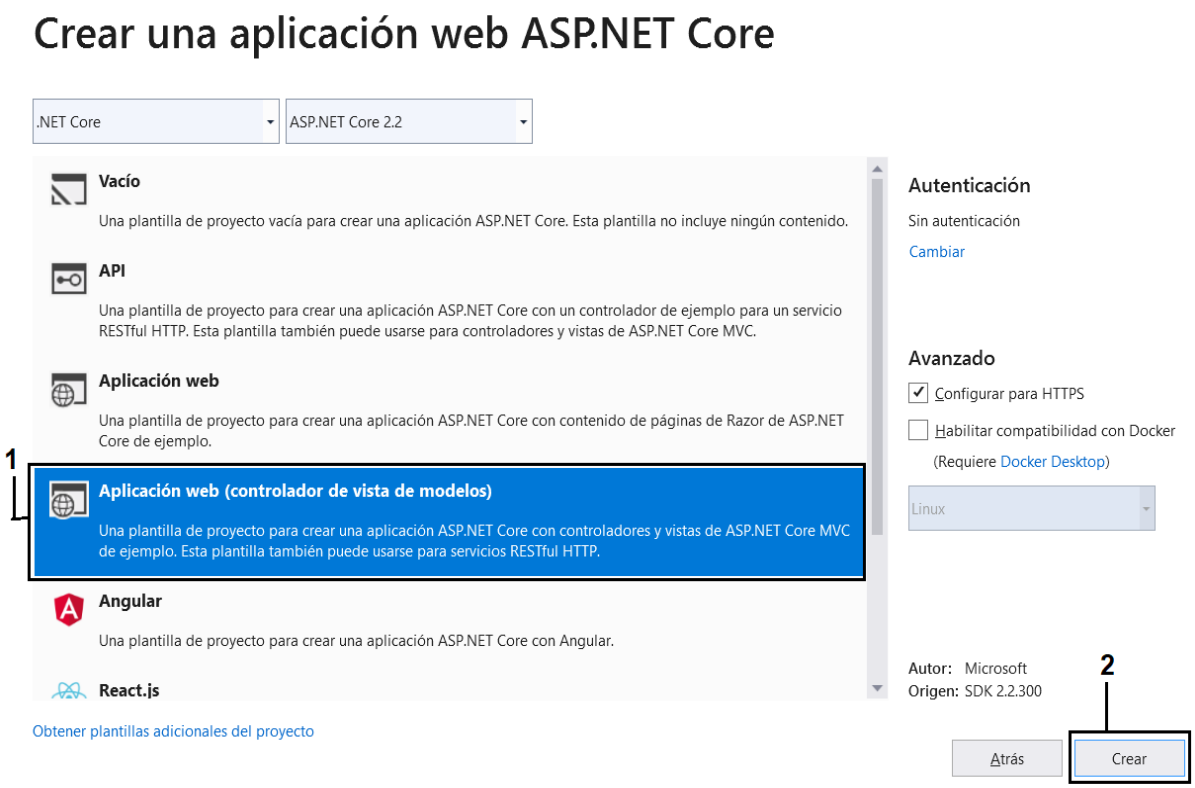


Figura 4.7.Ventana para la selección de la plantilla a trabajar.

Posterior a la instalación del SDK para la construcción de aplicaciones web con ASP.NET Core con el lenguaje C# y la creación de la instancia que contendrá el proyecto para el desarrollo de la plataforma, se procede a la construcción de la capa de acceso a datos e implementación de los patrones de unidad de trabajo y repositorio para la manipulación de datos.

4.3 Sprint 1: Implementación de la capa de acceso a datos

En la tabla 4.4 se muestra la planeación del Sprint 1, en este Sprint no se realizará ninguna historia de usuario del **product backlog** presentado en el capítulo 3. Para este Sprint se va a mostrar la creación de la capa de acceso a datos para la manipulación y almacenamiento de los datos de la plataforma propuesta. Donde los servicios proporcionados por esta capa serán utilizados durante el desarrollo de los módulos que abarcan las historias de usuario de la plataforma.

Tabla 4.4. Planeación del sprint 1.

Fecha de inicio	**Fecha de finalización**	**N° de semanas**
01-05-2019	30-06-2019	8

Actividades a realizar en el Sprint 1	**Fecha de inicio**	**Fecha de finalización**	**Horas de trabajo invertidas**
Creación de los modelos (clases)	01-05-2019	21-05-2019	168
Creación del contexto (clase para el establecimiento de la cadena de conexión)	22-05-2019	04-06-2019	112
Implementación del patrón unidad de trabajo	05-06-2019	19-06-2019	84
Implementación del patrón de repositorio	19-06-2019	30-05-2019	84
Total de horas de trabajo invertidas			448

La capa de acceso a datos contiene las clases que representan los **modelos** dentro de la **arquitectura MVC**, las cuales fueron creadas en base al diagrama de clases mostrado en el capítulo 3. Estos modelos son utilizados para la manipulación de información en tiempo de ejecución,

además estos modelos son una representación de las tablas de la base de datos, entonces esto permite la comunicación entre un sistema **orientado a objetos** y un **relacional.** Comúnmente cuando se desarrolla una aplicación que tenga la necesidad de almacenar información de forma permanente, el primer paso es la creación de una base de datos y posteriormente el desarrollo de la aplicación, la cual contendrá clases que representarán las tablas de la base de datos para almacenar información temporalmente (en tiempo de ejecución). Con **Entity Framework** a través de las clases creadas y sus relaciones permite la creación de la base de datos, este proceso se llama **migraciones** utilizando el enfoque **CodeFirst** como se mencionó en el capítulo 3, el mecanismo de las migraciones es similar a un sistema de control de versiones, en el cual se puede tener varias migraciones que representan las versiones de la base de datos, permitiendo poder regresar a una versión anterior en caso de que la actual no funcione correctamente, además cuando se trabaja con las migraciones no es necesario tener todas las clases para generar una base de datos, en caso de crear más clases o modelos se puede una nueva versión de la base de datos con base a los modelos actuales.

Para la implementación de la capa de acceso a datos es necesario instalar paquetes o librerías al proyecto que fue creado en Visual Studio (***PiScrum 2_0***) en la sección anterior (**Instalación del .NET Core SDK 2.2**).

4.3.1 Instalación de los paquetes en el proyecto PiScrum_2_0

Los paquetes que fueron instalados en la instancia del proyecto ***PiScrum 2_0*** se describen en la tabla 4.5.

Tabla 4.5. Descripción de los paquetes instalados en el proyecto PiScrum 2_0

Nombre del paquete	Uso	Versión
AutoMapper.Extensions. Microsoft.DependencyInjection	Este paquete permite hacer uso de la tecnología **AutoMapper**, la cual va a permitir el mapeo entre los **modelos** y los **ViewModels**.	6.0.0
Microsoft.EntityFrameworkCore. Tools	Este paquete proporciona **comandos** para realizar las **migraciones** con **EntityFramework** a través de una **consola** proporcionada por **Visual Studio**.	2.2.6
Npgsql	Es un proveedor ADO.NET (componentes del software .NET para el acceso a datos y a servicios de datos) de código abierto para **PostgreSQL**, el cual va a permitir el acceso al servidor de base de datos PostgreSQL.	4.1.2
Npgsql.EntityFrameworkCore. PostgreSQL	Estos dos paquetes (**PostgreSQL/Npgsql**) son proveedores para **EntityFramework Core**, es decir, permite hacer uso de las funciones de EntityFramework Core con el gestor de base de datos PostgreSQL.	2.2.4
Npgsql.EntityFrameworkCore. PostgreSQL.Design		1.1.0
System.Drawing.Common	Este paquete va a permitir la manipulación de los **bytes** de las **imágenes carga**das por los **usuarios** a través de la **plataforma**.	4.6.0
System.Linq.Dynamic.Core	Con este paquete se podrá realizar **consultas SQL nativas** con **LINQ** en **ASP.NET Core**.	1.0.19

A continuación, se describe como instalar un paquete en la instancia del proyecto ***PiScrum_2_0*** (ver indicador 1), la cual está dentro de la solución ***PiScrum_2_0*** (ver indicador 2). Para Instalar un paquete a un proyecto, es necesario hacer clic derecho en la instancia del proyecto y posteriormente aparecerá un cuadro de opciones, donde se tiene que seleccionar la opción **Administrar paquetes Nuget** o **NuGet** (ver indicador 3) (ver figura 4.8)

A continuación, se describe como instalar un paquete en la instancia del proyecto ***PiScrum_2_0*** (ver indicador 1), la cual está dentro de la solución ***PiScrum_2_0*** (ver indicador 2). Para Instalar un paquete a un proyecto, es necesario hacer clic derecho en la instancia del proyecto y posteriormente aparecerá un cuadro de opciones, donde se tiene que seleccionar la opción **Administrar paquetes Nuget** o **NuGet** (ver indicador 3) (ver figura 4.8)

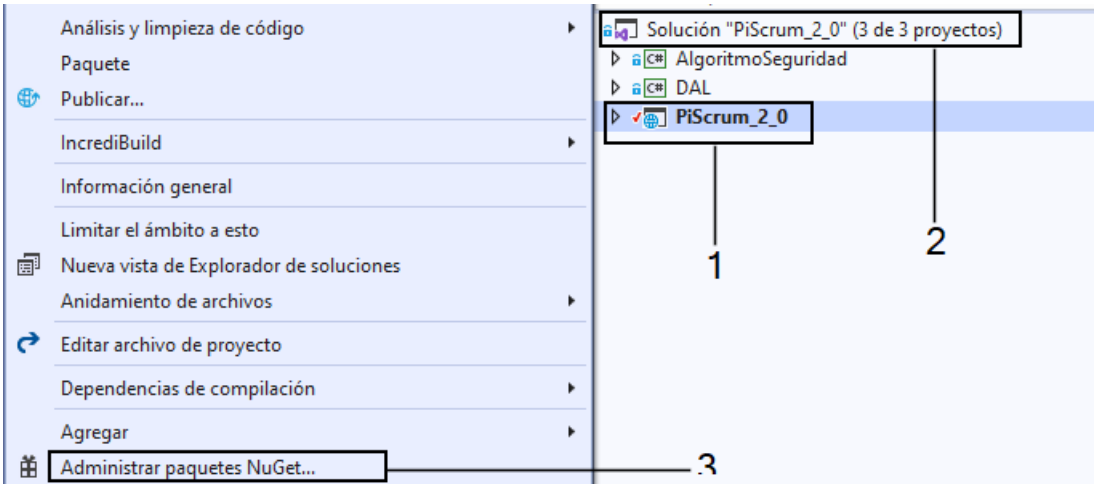


Figura 4.8. Proceso para instalación de paquetes a un proyecto.

Después de haber seleccionado la opción **Administrar paquetes NuGet**, aparece una pestaña con un formulario para realizar la búsqueda de paquetes como se ilustra en la figura 4.9. Primero se tiene que introducir el nombre del paquete en la caja de texto (ver indicador 1), si el administrador de paquetes encuentra la librería entonces aparecerá debajo de la caja de texto (ver indicador 2), al seleccionar la librería, en la parte derecha del **NuGet** se encuentra otro formulario donde se selecciona la versión de la librería a instalar (ver indicador 3) y por último se selecciona el botón **Instalar** (ver indicador 4).

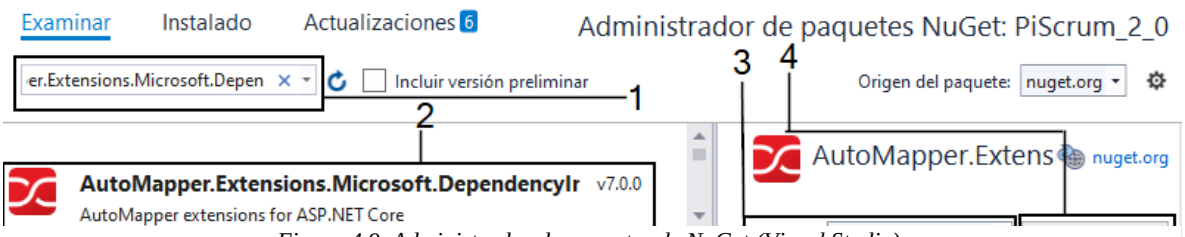


Figura 4.9. Administrador de paquetes de NuGet (Visual Studio).

Posteriormente al haber ver seleccionado el botón **Instalar**, aparecerá una ventana (ver figura 4.10) donde está indicando que paquetes (ver indicador 1) va a instalar a la instancia. Al seleccionar el botón **Aceptar** (ver indicador 2) indica a **NuGet** que acepta la instalación de los paquetes indicados.

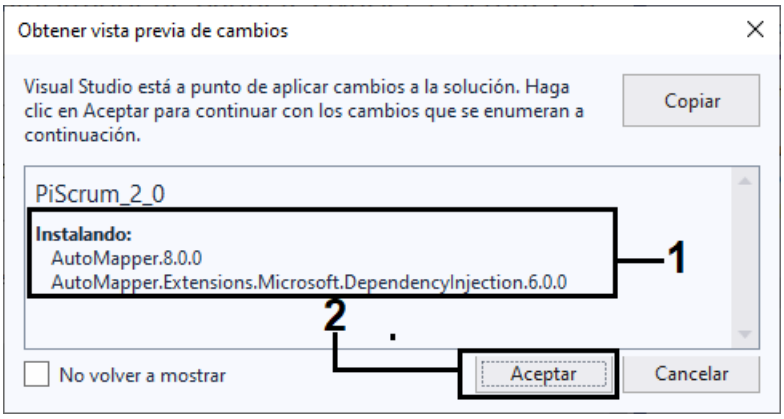


Figura 4.10. Ventana de confirmación para instalación de paquetes.

Este mismo proceso se realizó con los demás paquetes para su instalación, ya instalados los paquetes queda listo el proyecto ***PiScrum 2_0*** para el desarrollo de sus módulos.

Para manipular y almacenar la información de manera temporal y permanente, es necesario crear una capa de acceso a datos, la cual va a permitir comunicar las clases o modelos con las tablas de la base de datos y así poder manipular la información en la plataforma.

4.3.2 Creación de la capa de acceso a datos

Para la creación de esta capa es necesario agregar otra instancia para un proyecto, a la solución ***PiScrum 2_0*** para que posteriormente el nuevo proyecto y el proyecto ***PiScrum 2_0*** puedan comunicarse y este nuevo proyecto que será la capa de acceso a datos proporcione sus servicios al proyecto ***PiScrum 2_0***. Para agregar otra instancia a la solución es necesario hacer clic derecho (ver indicador 1), entonces se mostrará un cuadro de opciones (ver indicador 2), donde se selecciona la opción **Agregar**, en seguida **Nuevo proyecto** (ver indicador 3) (ver figura 4.11).

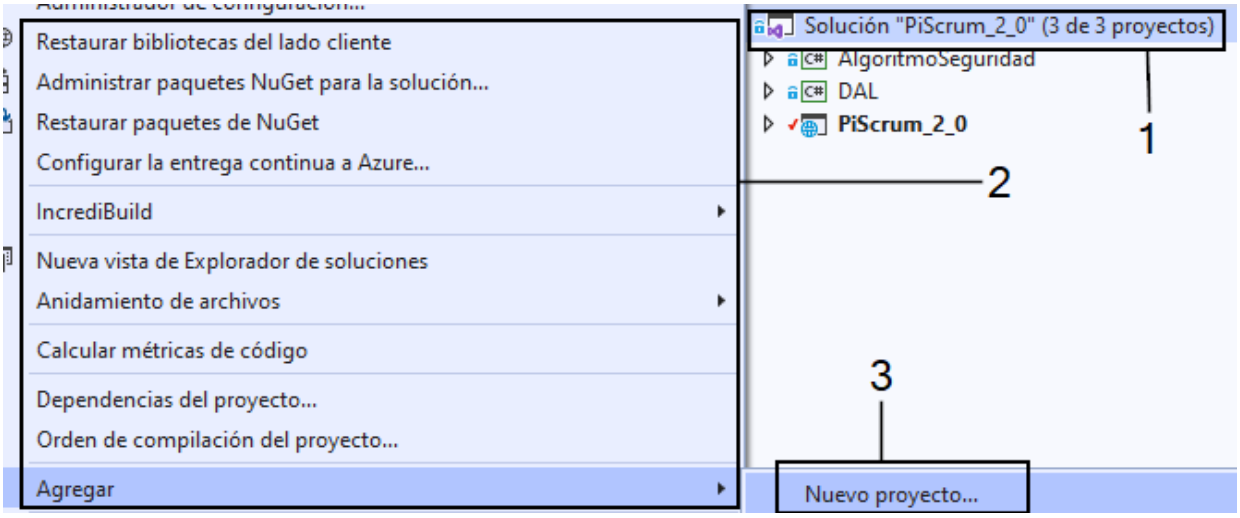


Figura 4.11. Proceso para agregar un nuevo proyecto a la solución.

Al momento de seleccionar la opción **Nuevo proyecto**, aparecerá una ventana que proporciona diferentes tipos de proyecto. Para la creación de la capa de acceso a datos es necesario una **biblioteca de clases (.NET Core)** (ver indicador 1) (ver figura 4.12) el proceso es similar para la creación de un proyecto como se expuso anteriormente, cuando se creó la instancia del proyecto ***PiScrum 2_0***.

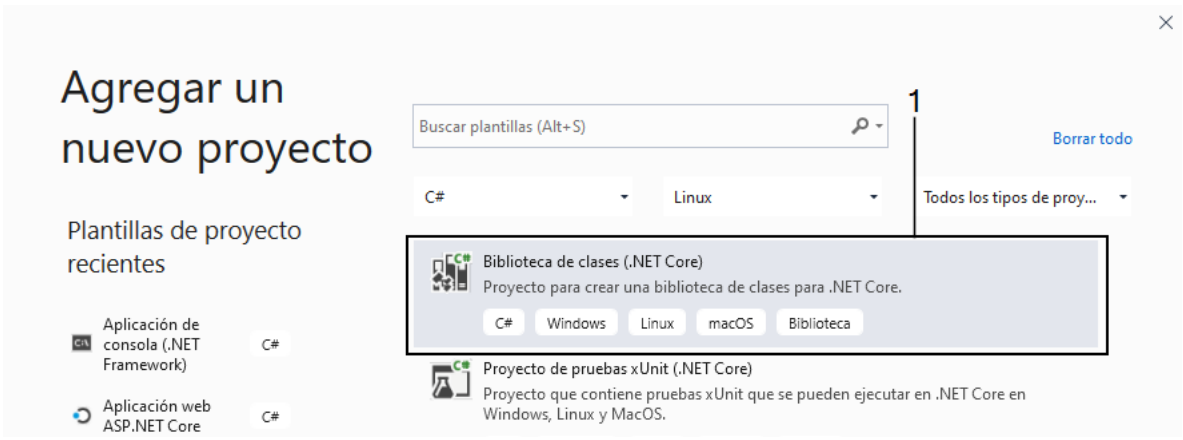


Figura 4.12. Asistente de Visual Studio para agregar un nuevo proyecto.

El proyecto que será la capa de acceso a datos se le asignó el nombre de **DAL** (Data Access Layer, en español Capa de Acceso a Datos) (ver indicador 1) como se observa en la figura 4.13.

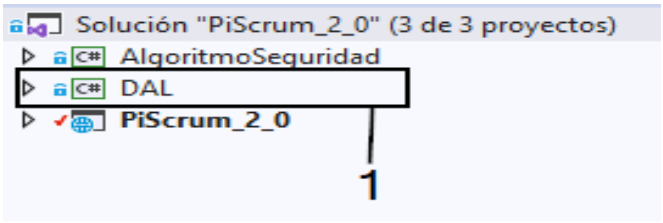


Figura 4.13. Solución que contiene los proyectos PiScrum, DAL y AlgoritmoSeguridad.

En el proyecto **DAL** se crearon las clases con base al diagrama de clases presentado en el capítulo 3, como se puede observar en la parte derecha del IDE (ver indicador 1) (ver figura 4.14), en la parte izquierda del IDE se encuentran las propiedades (ver indicador 2) de la clase **Tarea** (archivo con extensión **.cs**, ver indicador 3).

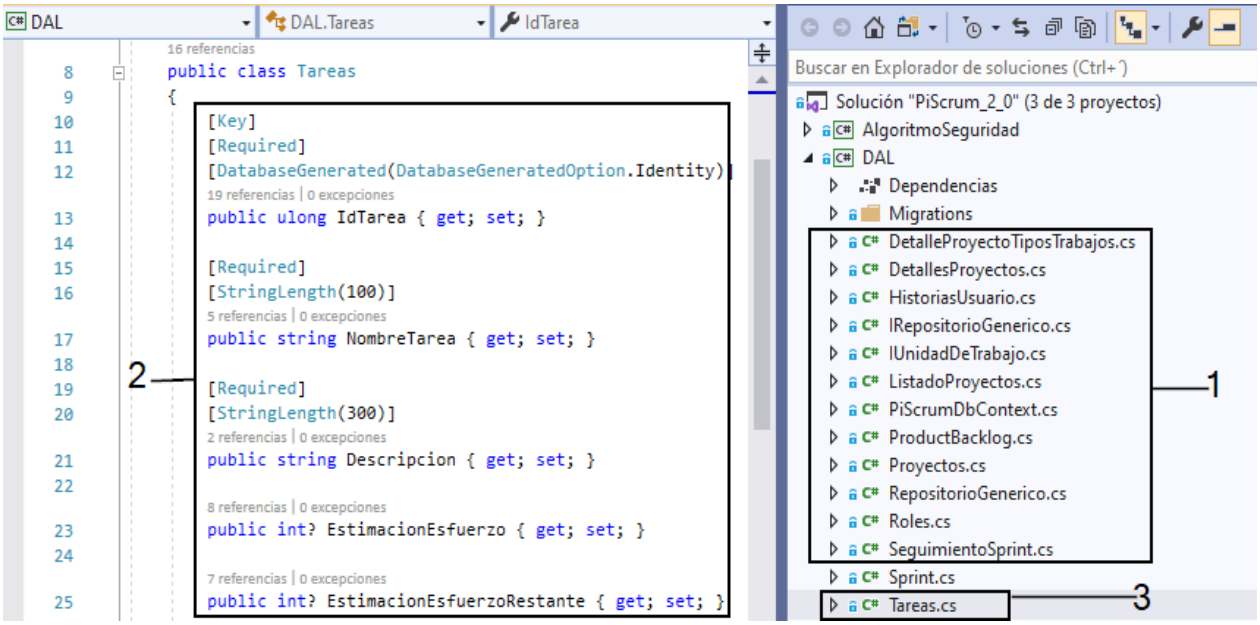


Figura 4.14. La clase Tareas y sus propiedades en C#.

Con C# a través de los espacios de nombres **System.ComponentModel.DataAnnotations** y **System.ComponentModel.DataAnnotations.Shema** (ver indicador 1) permiten establecer **clases de atributos** (**DataAnnotations**) para definir metadatos (ver indicador 2) para los controles de datos en **ASP.NET Core MVC** (ver figura 4.15).

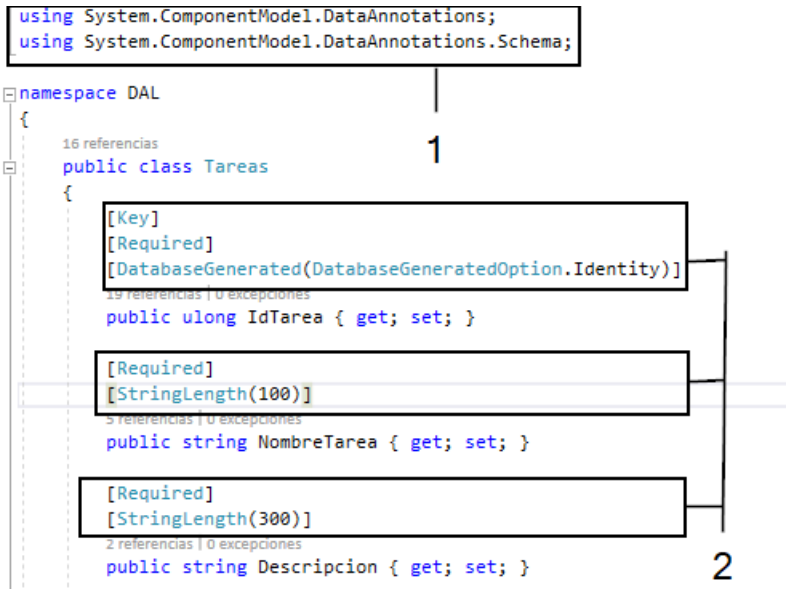


Figura 4.15. DataAnnotations de la clase Tareas.

Al establecer estos metadatos a las propiedades de la clase ***Tareas*** permitirá que, al momento de hacer las **migraciones** para generar la **base de datos**, los campos de la tabla ***Tareas*** tendrá estos mismos metadatos de la clase ***Tareas***, por ejemplo, estableciendo una longitud máxima en una cadena de caracteres. También al establecer **DataAnnotations** a las propiedades de una clase permitirán realizar **validaciones** en el **Front-End** y **Back-End**.

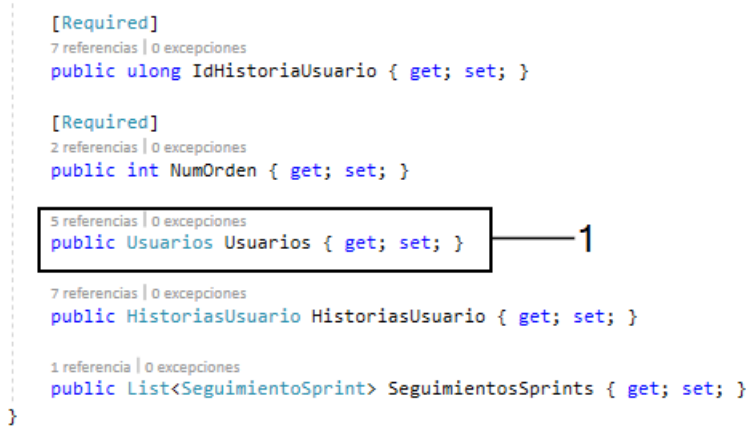


Figura 4.16. Propiedad de navegación para la clase Usuarios.

Para la creación de las relaciones entre las tablas, es necesario crear propiedades de navegación en las clases, las cuales hacen referencia a otra clase. En la clase ***Tareas*** se creó una **propiedad de**

navegación (ver indicador 1) que hace referencia a la clase ***Usuarios*** (ver diagrama de clases del capítulo 3) indicando que una tarea sólo le pertenece a un usuario como se ilustra en la figura 4.16.

Por otro lado, la clase ***Usuarios*** tiene una propiedad de navegación de tipo lista, la cual a su vez es de tipo ***Tareas*** (ver indicador 1 de la figura 4.17), indicando que un usuario puede tener varias tareas asignadas. Esto se realizó debido a que a cuando un usuario en la plataforma sea del equipo de desarrollo, podrá asignarse varias tareas.

```
public class Usuarios
{
    [Key]
    [Required]
    [StringLength(13)]
    9 referencias | 0 excepciones
    public string RfcUsuario { get; set; }

    [Required]
    [StringLength(70)]
    4 referencias | 0 excepciones
    public string Nombre { get; set; }

    [Required]                                          1
    [StringLength(70)]
    4 referencias | 0 excepciones
    public string ApellidoP { get; set; }
    //Relaciones
    1 referencia | 0 excepciones
    public List<Tareas> Tareas { get; set; }
}
```

Figura 4.17. Clase Usuarios con su relación entre la clase Tareas.

Para la creación de la base de datos es implementó una clase llamada ***PiScrumDbContext*** (ver indicador 1) (ver figura 4.18) la cual hereda de la clase **DbContext** (ver indicador 2) (clase proporcionada por C#) que provee una instancia que representa una sesión con la base de datos y se puedan realizar consultar y guardar instancias de las entidades (clases), la cadena de conexión se inyecta a través del segundo constructor (ver indicador 3) de la clase ***PiScrumDbContext***. El objetivo de la clase ***PiScrumDbContext***, es la creación de entidades a través de la implementación de propiedades de tipo **DbSet** (ver indicador 4) con las clases que se muestran en la figura 4.14 (ver indicador 1) en la clase ***PiScrumDbContext***. Al momento de crear entidades con las clases, permitirá guardar instancias de estas entidades, guardado datos en la base de datos. Con este

proceso de crear entidades, también permite establecer que clases se convertirán en tablas al momento se hacer las migraciones para la generación de la base de datos.

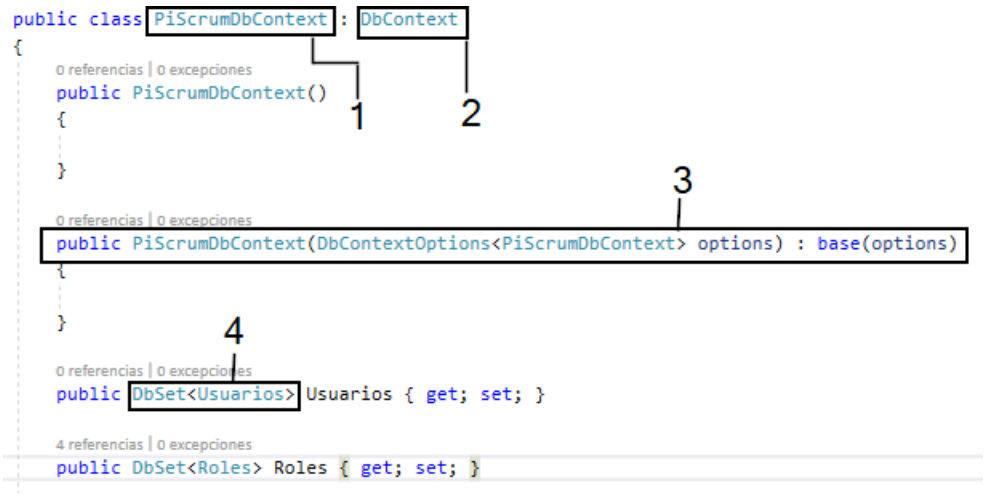


Figura 4.18. Clase PiScrumDbContext donde se crearon las entidades y relaciones.

En la clase ***PiScrumDbContext*** a través de la herencia de **DbContext** se tiene acceso al método **OnModelCreating** (ver indicador 1, figura 4.19), permitiendo la creación de las relaciones entre las tablas a través de las propiedades de navegación de las clases, utilizando **Fluent API** (forma avanzada para especificar la configuración del modelo).

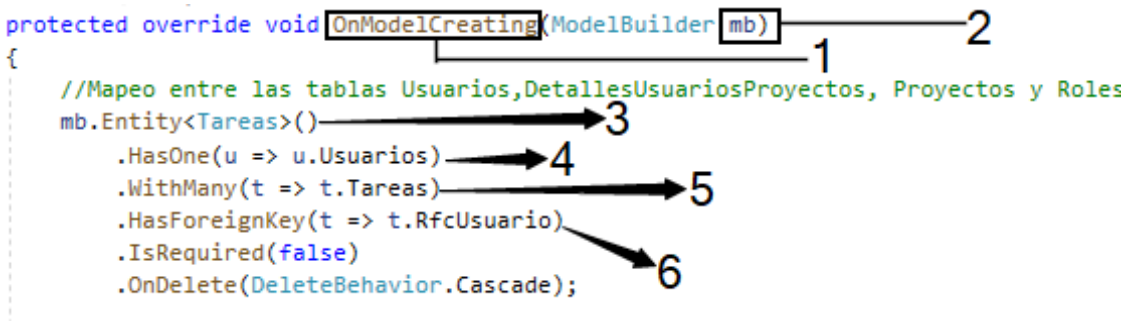


Figura 4.19. Creación de las relaciones con FLUENT API.

Con el parámetro ***mb*** (ver indicador 2) de tipo **ModelBuilder** permite usar **Fluent API**. A través del parámetro ***mb*** se indica a que modelo se realizarán las configuraciones (ver indicador 3) con **Fluent API**. Con los métodos **HasOne** (ver indicador 4) y **WithMany** (ver indicador 5), donde el primero se le envía como argumento la propiedad de navegación creada en la clase ***Tareas*** (ver indicador 4) y el segundo método se le envía la propiedad de navegación creada en la clase

Usuarios, al realizar esto se creará una relación de uno a muchos entre las tablas ***Usuarios*** y ***Tareas*** al momento de crear la base de datos. Para establecer que propiedad será llave foránea en la clase dependiente, se utiliza el método **HasForeignKey** (ver indicador 6) donde se le envía como argumento la propiedad ***RfcUsuario*** creada en la clase ***Tareas*** (clase dependiente). Este mismo proceso se realizó con las demás clases para la normalización de la base de datos que almacenará la información de la plataforma.

Para que al momento de realizar las migraciones es necesario establecer una cadena conexión que apunte a un servidor una base de datos, para ello se implementa la cadena de conexión en un archivo llamado **appsetting.json** proporcionado por **ASP.NET Core,** donde el establecimiento de la cadena de conexión (ver indicador 1) se realiza a través del formato Notación de Objeto JavaScript (en inglés JavaScript Object Notation o JSON) como se observa en la figura 4.20.

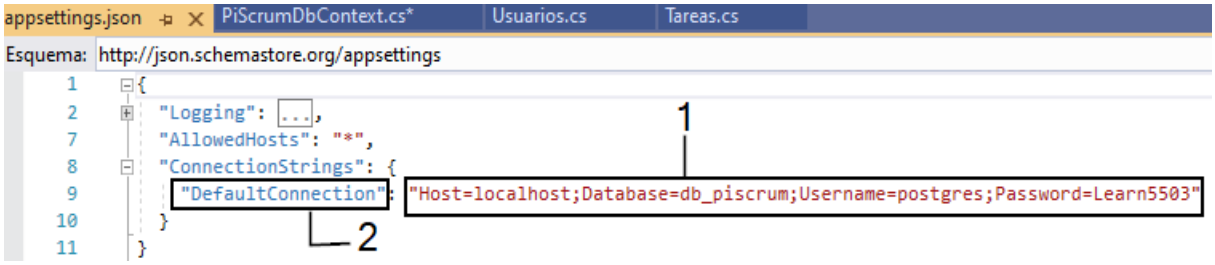


Figura 4.20. Establecimiento de la cadena de conexión.

A esta cadena de conexión se le proporcionó un nombre (**DefaultConnection**) (ver indicador, figura 4.20) para ser obtenida por un método proporcionado por **ASP.NET Core** llamado **GetConnectionString** (ver indicador 2), este método es implementado en una clase llamada **Startup** (ver indicador 1) proporcionada por **ASP.NET Core** (ver figura 4.21). La clase **Startup** se encuentra en el proyecto ***PiScrum 2_0***, en esta clase se registran todos los servicios necesarios para la plataforma y se registran aquí debido a que esta clase se manda a llamar primero cuando se ejecuta la aplicación inyectado todos los servicios para ser utilizados durante el ciclo de vida de la plataforma.

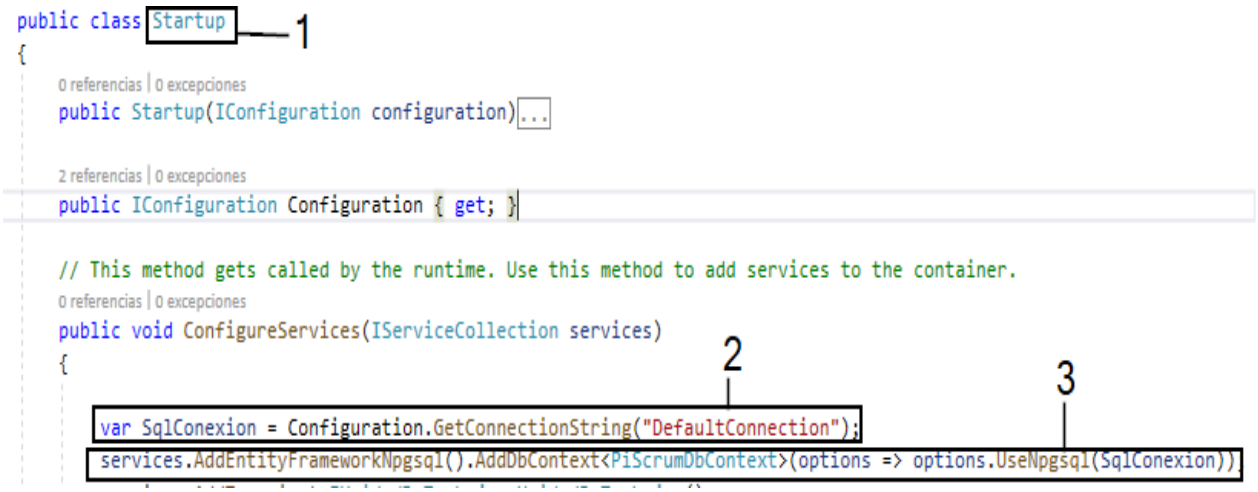


Figura 4.21. Registros del contexto e inyección de la cadena de conexión en el Startup.

Cuando el método **GetConnectionString** obtiene la cadena de conexión es almacenada en una variable llamada **SqlConexion** donde se inyecta a través del método **UseNpgsql** (ver indicador 3) para establecer la conexión con **PostgreSQL**. Para permitir la manipulación de datos en la plataforma, es necesario registrar un servicio de tipo contexto que se hace a través del método **AddContext** (ver indicador 3) proporcionado por la librería **EntityFrameworkNpgsql**.

Con la implementación de los modelos, relaciones y el registro del servicio de tipo contexto de la clase ***PiScrumDbContext*** en la clase **Startup**, se procede a realizar la migración para la creación de la base de datos en el gestor de base de datos **PostgreSQL** con las configuraciones expuestas. Para realizar la migración solamente es necesario de dos comandos proporcionados por el paquete Microsoft.EntityFrameworkCore.Tools, estos comandos son **add-migration** <nombre de la migración> y **update-database** <nombre de la migración creada en caso de existir más de una migración> para introducir estos comandos se hacen a través de la **consola del administrador de paquetes** (ver indicador 1) (ver figura 4.22), cuando se ejecuta el comando **add-migration Primer_Migracion** (ver indicador 2) envía un mensaje donde inicializa el contexto usando el proveedor **Npgsql.EntityFrameworkCore.PostgreSQL** (paquete instalado), después de haberse

ejecutado el comando para crear la migración, en el proyecto **DAL** se crea una carpeta automáticamente llamada **Migrations** (ver indicador 3) la cual contiene los archivos necesarios de las configuraciones realizadas con los modelos para la creación de la base de datos.

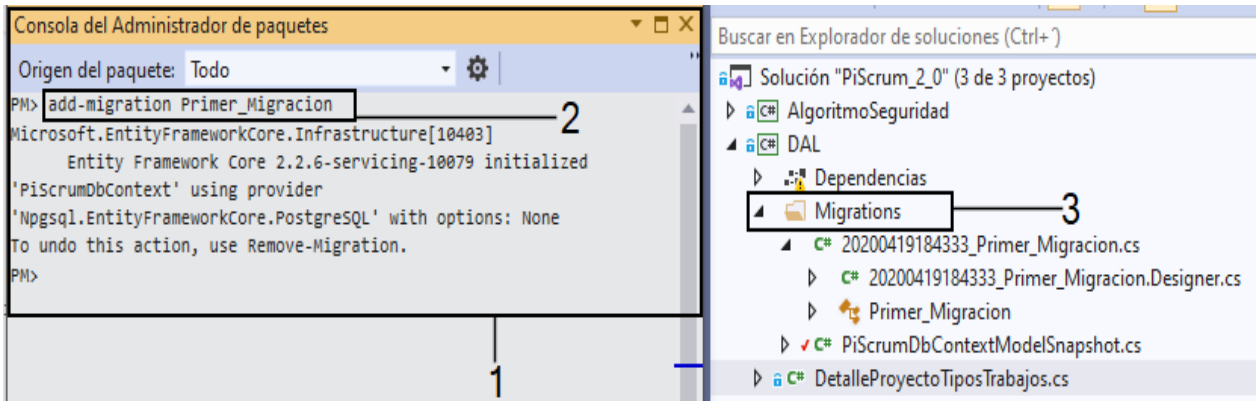


Figura 4.22. Ejecución del comando add-migration para crear una migración.

Para generar la base de datos con las configuraciones cargadas con la migración es necesario ejecutar el comando **update-database** en la **consola del administrador de paquetes** donde se mostrará el código SQL, en la figura 4.23 se ilustra la creación de la tabla ***Tareas*** con sus respectivos campos con base a las propiedades de la clase ***Tareas*** (ver figura 4.14 e indicador 2).

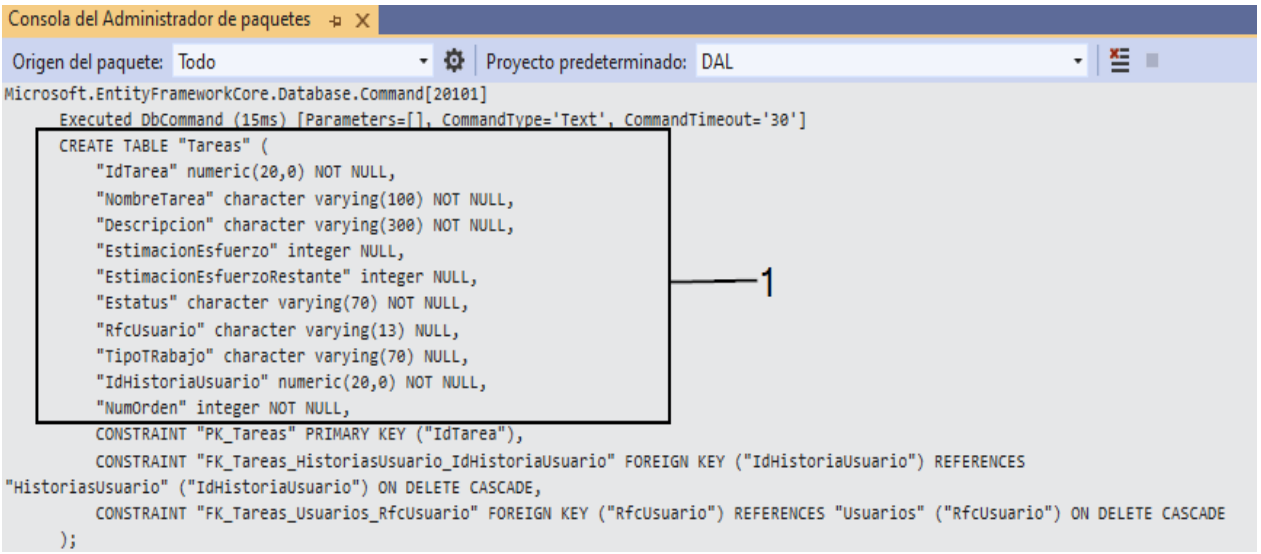


Figura 4.23. Ejecución del comando update-database para la generación de la base de datos.

Después se crear la base de datos con la migración, en la figura 4.24 se muestra una comparativa de las tablas creadas en **PostgreSQL** (ver indicador 1) y las clases implementadas en la capa de acceso a datos contenida en el proyecto **DAL** (ver indicador 2), donde los modelos corresponden

con las tablas de la base de datos con nombre ***db_piscrum*** (ver indicador 3) (nombre asignado en la cadena de conexión, ver figura 4.20).

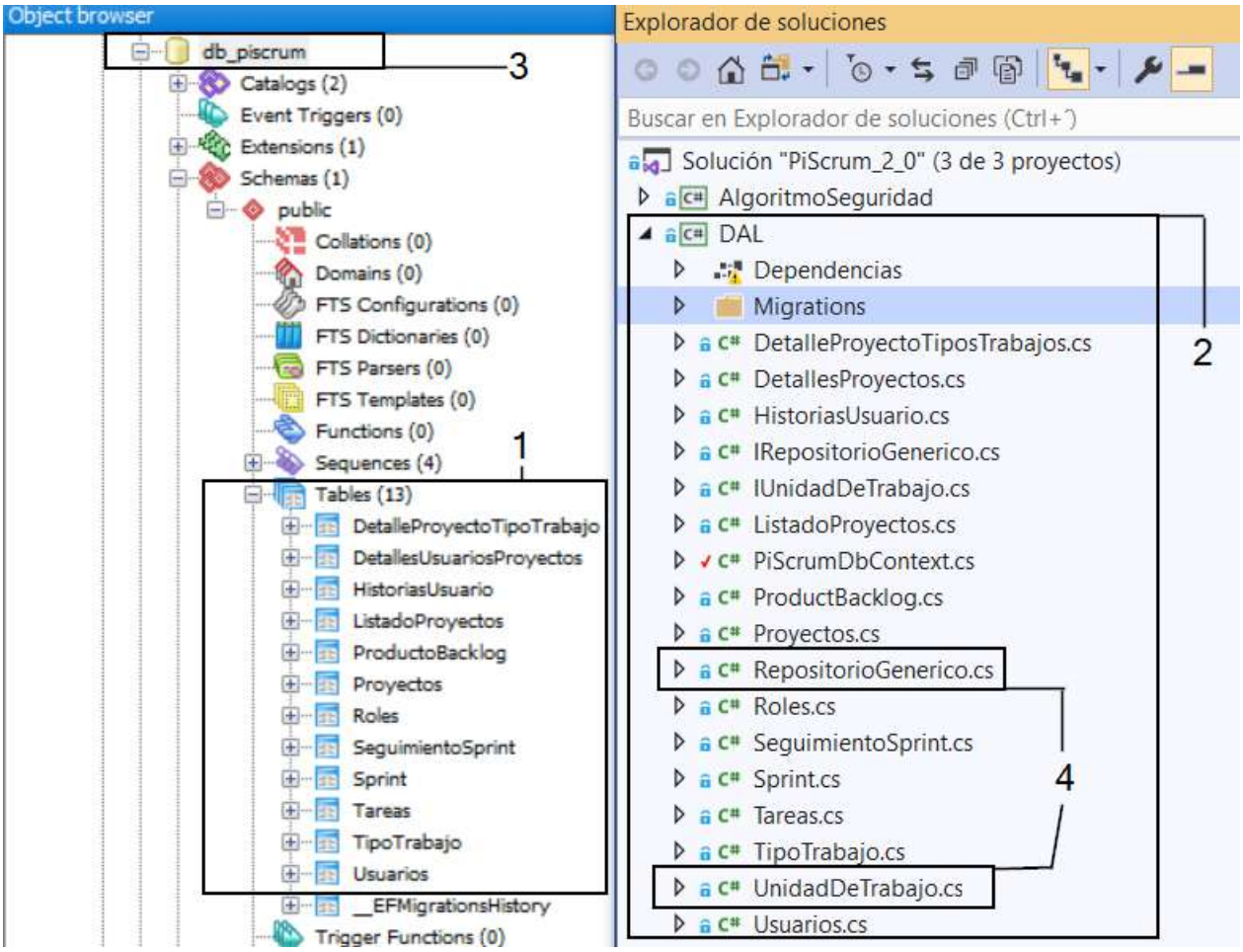


Figura 4.24. Comparación de las clases con las tablas creadas en la base de datos posterior a la migración.

Para tener una infraestructura que permita y asegure la manipulación de los datos al momento realizar una transacción a través de la plataforma, se implementaron dos clases ***UnidadDeTrabajo*** y ***RepositorioGenerico*** (ver indicador 4) en la capa de acceso a datos como se ilustra en la figura 4.24. Estas dos clases corresponde a los patrones unidad de trabajo y repositorio, en la clase ***UnidadDeTrabajo*** se crearon campos inyectado interfaces ***IRepositorioGenerico*** de tipo **entity** (son entidades como: Tareas, Usuarios, etc) (ver indicador 1 en la figura 4.25), de esta forma no es necesario tener un repositorio por cada entidad (clase o modelo), permitiendo la reutilización de código al momento de agregar más modelos a la capa de acceso a datos.

```
public class UnidadDeTrabajo : IUnidadDeTrabajo
{
    private readonly PiScrumDbContext db;

    0 referencias | 0 excepciones
    public UnidadDeTrabajo(PiScrumDbContext db)
    {
        this.db = db;
    }

    private IRepositorioGenerico<ListadoProyectos> ListadoProyectos;
    private IRepositorioGenerico<Usuarios> Usuarios;
    private IRepositorioGenerico<DetallesProyectos> DetallesProyectos;
    private IRepositorioGenerico<Proyectos> Proyectos;
    private IRepositorioGenerico<ProductBacklog> ProductBacklog;
    private IRepositorioGenerico<HistoriasUsuario> HistoriasUsuario;
    private IRepositorioGenerico<DetalleProyectoTiposTrabajos> DetalleProyectoTiposTrabajos;
    private IRepositorioGenerico<TipoTrabajo> TipoTrabajo;
    private IRepositorioGenerico<Sprint> Sprint;
    private IRepositorioGenerico<Tareas> Tareas;
    private IRepositorioGenerico<SeguimientoSprint> SeguimientoSprint;
```

1

Figura 4.26. Campos para la manipulación de datos de las entidades.

```
public class UnidadDeTrabajo : IUnidadDeTrabajo
{
    private readonly PiScrumDbContext db;

    0 referencias | 0 excepciones
    public UnidadDeTrabajo(PiScrumDbContext db)
    {
        this.db = db;
    }
    private IRepositorioGenerico<Tareas> Tareas;
    private IRepositorioGenerico<SeguimientoSprint> SeguimientoSprint;

    24 referencias | 0 excepciones
    public IRepositorioGenerico<Tareas> TareasRepositorio
    {
        get
        {
            return Tareas = Tareas ?? new RepositorioGenerico<Tareas>(db);
        }
    }
    4 referencias | 0 excepciones
    public async Task<int> SaveAsync()
    {
        return await db.SaveChangesAsync();
    }
```

2

3

1

Figura 4.25. Propiedades de las entidades para realizar transacciones en la base de datos.

En la misma clase (***UnidadDeTrabajo***) se crearon propiedades (ver indicador 1 en la figura 4.26) para realizar transacciones en la base de datos con las entidades proporcionadas por los campos implementados (ver figura 4.25), para ello necesitan de una conexión a la base de datos (contexto) que es proporcionada por el campo ***db*** (ver indicador 2 en la figura 4.26) de tipo

PiScrumDbContext donde obtiene el servicio (de tipo contexto) inyectado a través del constructor ***UnidadDeTrabajo*** (ver indicador 3 en la figura 4.26).

Para realizar operaciones con las propiedades proporcionadas por el patrón ***UnidadDeTrabajo*** se implementaron las operaciones básicas en la clase ***RepositorioGenerico***, que se realizan en las tablas de una base de datos las cuales son **Agregar**, **Consultar**, **Modificar** y **Eliminar** (por sus siglas en inglés, CRUD) cada una de estas operaciones pueden ser implementadas con cualquier entidad, debido a que esta clase es genérica de tipo **entity** (entidad). En la figura 4.27 se pueden observar los métodos (ver indicador 1) que proporcionan el mecanismo para realizar las transacciones en la base de datos (***db_piscrum***) de la plataforma propuesta. Para realizar estas transacciones en la base de datos es necesario de un contexto, el cual es proporcionado a través del constructor ***RepositorioGenerico*** (ver indicador 2).

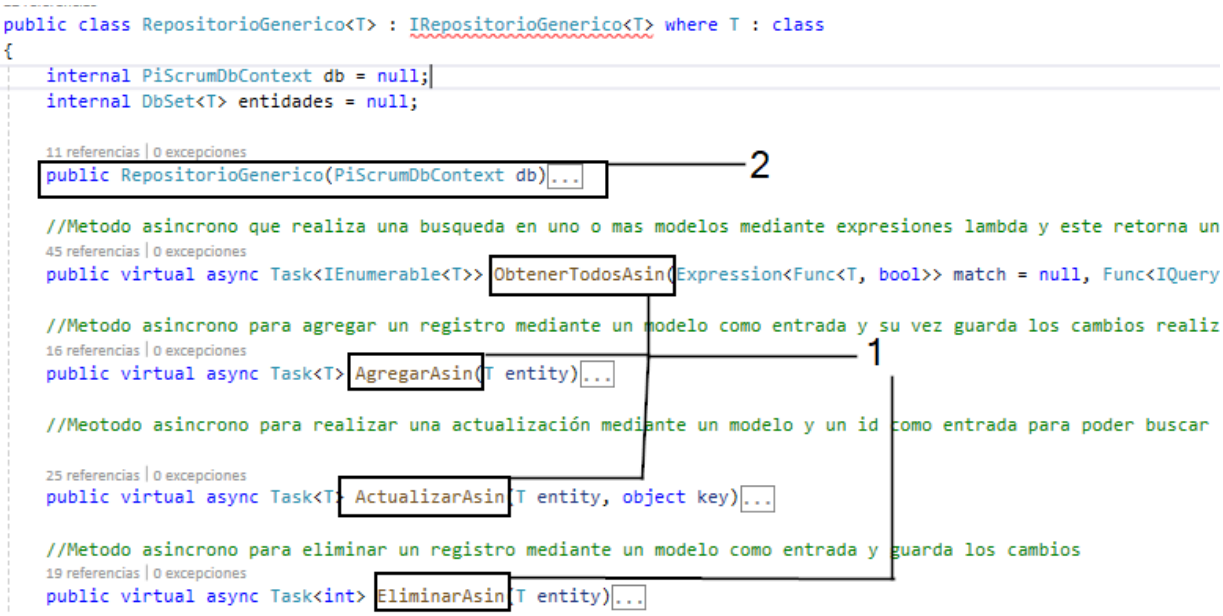


Figura 4.27. Implementación del patrón RepositorioGenerico.

En los siguientes apartados se presentarán los Sprints donde se mostrará el desarrollo de las funciones que componen la plataforma, estas funciones requerirán de los modelos de la capa de acceso a datos y los mecanismos proporcionados por las clases ***UnidadDeTrabajo*** y

RepositorioGenerico, para realizar operaciones con ellos y para realizar cambios en la base de datos.

4.4 Sprint 2: Autenticación, autorización de usuarios y creación de proyectos

En la tabla 4.6 se presenta la planeación del Sprint 2, donde cada historia de usuario se desglosa en tareas que tienen un tiempo estimado para su realización, así como el inicio y finalización de este Sprint.

Tabla 4.6. Planeación del Sprint 2.

Fecha de inicio		**Fecha de finalización**		**N° semanas**	
01-07-2019		03-08-2019		5	
Historia de usuario	**Tareas**	**Fecha inicio**	**Fecha de finalización**	**Horas de trabajo de la tarea**	**Horas de trabajo total de la historia de usuario**
HU 1	Crear una función para registrar los usuarios en la plataforma.	01-07-2019	04-07-2019	32	**56**
	Crear una función que valide si el usuario está registrado.	05-07-2019	05-07-2019	8	
	Crear la vista para que el usuario pueda registrarse en la plataforma.	06-07-2019	07-07-2019	16	
	Implementar una función para enviar un correo electrónico.	08-07-2019	08-07-2019	**8**	

HU 2	Desarrollar un método para que el usuario realice una confirmación con el correo electrónico registrado.	09-07-2019	09-07-2019	**8**	**28**
	Desarrollar una vista para que a través de esta el usuario confirme su correo electrónico.	10-07-2019	10-07-2019	**5**	
	Crear un método para validar si el usuario ha confirmado su correo electrónico.	10-07-2019	11-07-2019	**4**	
	Crear una función para validar si el correo electrónico introducido existe la base de datos de la plataforma.	11-07-2019	11-07-2019	**3**	
HU 3	Implementar un método para crear un nuevo proyecto en la plataforma	11-07-2019	17-07-2019	**60**	**112**
	Desarrollar una función que los usuarios involucrados en el proyecto pueda consultar su información.	18-07-2019	20-07-2019	**20**	
	Crear una función que solamente el usuario con rol de **product owner** pueda	20-07-2019	24-07-2019	**32**	

	actualizar información del proyecto.				
HU 4	Desarrollar un método que permita al usuario con rol de **product owner** agregar colaboradores a un proyecto.	24-07-2019	28-07-2019	**32**	**84**
	Implementar una función que permita la consulta de los colaboradores en un proyecto.	28-07-2019	29-07-2019	**10**	
	Crear una función para dar o denegar acceso a los colaboradores en un proyecto.	29-07-2019	31-07-2019	**12**	
	Crear un formulario en una vista para agregar un colaborador a un proyecto.	31-07-2019	01-08-2019	**10**	
	Crear una vista para consultar la lista de colaboradores en un proyecto.	01-08-2019	03-08-2019	**20**	
		Total de horas de trabajo invertidas			**280**

Cada una de estas historias de usuario con sus tareas serán abarcadas en el presente desarrollo de este Sprint, el cual contempla la implementación de la autenticación, autorización y creación de

los proyectos en la plataforma. En este apartado el desarrollo será presentados en dos fases **Back-End** y **Front-End**.

4.4.1 Back-End: Registro de usuarios en la plataforma

Como se mencionó en el capítulo 2, la arquitectura que será implementada en la plataforma será MVC. ASP.NET Core provee una forma de crear aplicaciones web con base al patrón arquitectónico MVC. En la figura 4.28 se puede observar la implementación de una clase llamada ***UsuariosController*** (ver indicador 1) que hereda de la clase **Controller** (ver indicador 2), la cual proporciona la forma de trabajar con MVC. Se creó este controlador (***UsuariosController***) para implementar la lógica de negocio para la gestión de usuarios (creación, cambio de contraseña consulta y modificación de información, etc). Así como las clases ***UnidadDeTrabajo*** y ***RepositorioGenerico*** requieren del servicio de tipo contexto (***PiScrumDbContext***) para realizar transacciones, en cada controlador se requiere de este tipo de servicio y es inyectado a través de sus constructores. Además de este servicio, se inyectan los servicios para implementar los patrones unidad de trabajo, repositorio (***UnidadDeTrabajo*** y ***RepositorioGenerico***), mapeo entre clases (**AutoMapper**) y otro servicio que permite proveer información del **web host** (ver indicador 3). Cada uno de estos servicios serán utilizados en la implementación de la lógica de negocio contenida en cada acción (método) del controlador ***UsuariosController***.

```
public class UsuariosController : Controller
{
    private readonly IHostingEnvironment he;
    private readonly IUnidadDeTrabajo unidadDeTrabajo;
    private PiScrumDbContext db;
    private readonly IMapper mapper;

    0 referencias | 0 excepciones
    public UsuariosController(IUnidadDeTrabajo uofw, PiScrumDbContext _db, IMapper _mapper, IHostingEnvironment e)
    {
        unidadDeTrabajo = uofw;
        db = _db;
        mapper = _mapper;
        he = e;
    }
```

Figura 4.28. Implementación del controlador para la lógica de negocio de los usuarios.

Para la implementación de la lógica de negocio en un controlador es necesario crear acciones (métodos), estas acciones contendrán esa lógica de negocio. En la figura 4.29 se observan dos acciones con el mismo nombre (***Registrar***) (ver indicador 1), donde primero responde al método **GET** y este se ejecuta cuando un usuario solicita la vista correspondiente para registrarse en la plataforma y el segundo responde al método **POST** (ver indicador 2) y este se ejecuta cuando el usuario envía la información a través de un formulario por este mismo método (**POST**) al servidor.

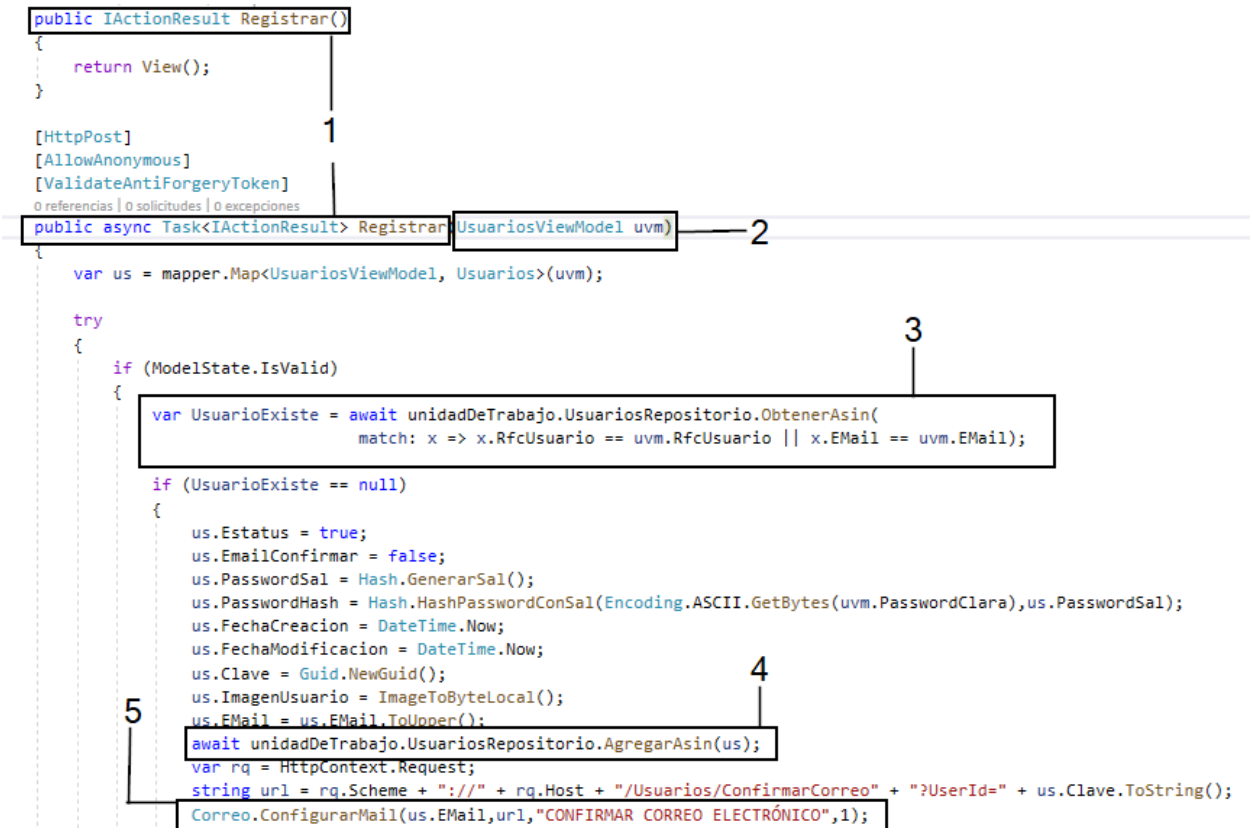


Figura 4.29. Acción Registrar para el registro de usuarios.

Cuando el usuario envía la información desde la vista correspondiente a la acción ***Registrar***, en el método **POST** recibe como parámetro un **ViewModel** llamado ***uvm*** de tipo ***UsuariosViewModel*** (indicador 3), el cual contiene la información proporcionada por usuario que introduce en la vista. La información contenida en ***uvm*** es procesada dentro de la acción ***Registrar*** de tipo **POST**, donde primero se verifica si existe un usuario con el correo electrónico o rfc introducido (ver indicador 3), en caso de no existir procede al registro de la información en la base de datos, utilizando los

servicios proporcionados las clases ***UnidadDeTrabajo*** y ***RepositorioGenerico***, permitiendo implementar el método ***AgregarAsin*** (ver indicador 4), el cual recibe como argumento un modelo (**us**). Por ultimo después de realizar el registro en la base de datos, la plataforma envía un correo electrónico a la dirección (correo electrónico) proporcionada por el usuario para su validación, este proceso lo realiza el método ***ConfigurarMail*** (ver indicador 5).

4.4.2 Back-End: Acceso a la plataforma y creación de proyectos

Para hacer un inicio de sesión se necesita agregar un servicio llamado **AddAuthentication** (ver indicador 1) a la clase **Startup** donde se configura una ruta (ver indicador 2) para que cuando un usuario quiera acceder a un área de la plataforma y esta esté restringida y no tenga las credenciales correctas lo envié a una vista que contenga un formulario para que inicie sesión. Otra configuración que se tiene que realizar es la creación de una cookie (***PSCookieAutenticationSheme***) (ver indicador 3) para que se almacene la información encriptada de la sesión (ver figura 4.30).

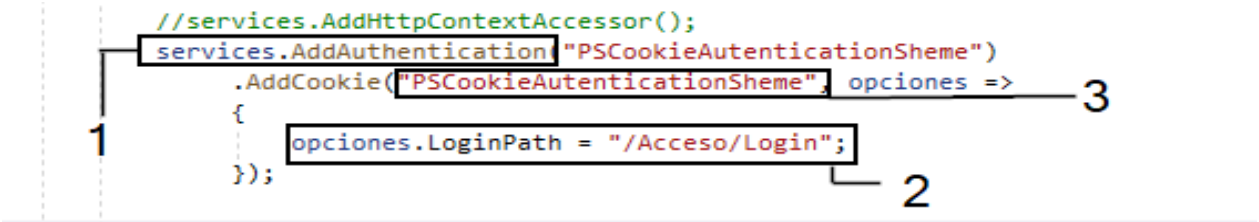


Figura 4.30. Configuración y creación de la cookie.

Con la configuración de la cookie, puede hacerse uso de ella para la implementación del inicio de sesión para los usuarios, de la misma forma que en la acción ***Registrar*** del controlador ***UsuariosController,*** se implementó otro controlador llamado ***AccesoController*** (ver indicador 1) con dos acciones tipo GET (ver figura4.31) y POST (ver figura 4.32) para iniciar sesión.

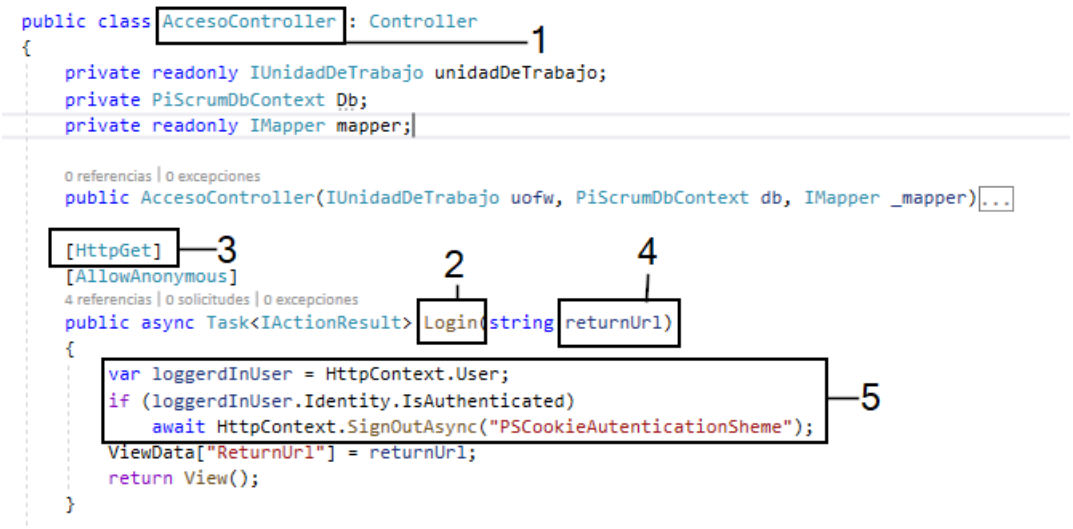


Figura 4.31. Implementación de la acción Login (GET) para iniciar sesión.

En las líneas de código que se observan en la figura 4.31 está la implementación de la acción ***Login*** (ver indicador 2), donde el método GET (ver indicador 3) recibe como parámetro (***returnUrl***) (ver indicador 4) una cadena que contendrá una url, esto es siempre y cuando un usuario que no haya iniciado sesión e intente acceder a un área restringida, la plataforma solicitará que primero inicie sesión re direccionándolo a la url configurada en el servicio ***AddAuthetication*** (ver figura 4.30, indicador 2) y posteriormente al haber iniciado sesión lo re direccionará a la url solicitada. En caso de haber una sesión activa se destruye la cookie (ver indicador 5).

Para que el usuario pueda acceder a través de sus credenciales (correo electrónico y contraseña), se creó el método **POST** en la acción ***Login*** (ver figura 4.32), donde este método recibe las credenciales introducidas por el usuario a través del parámetro ***model*** de tipo ***LoginViewModel*** (ver indicador 1).

Figura 4.32. Creación de la Cookie con los datos del usuario para iniciar sesión.

Estos datos (credenciales del usuario) son procesados dentro de la acción, donde primero se valida si existe un usuario con esas credenciales en la base de datos (ver indicador 2), en caso de existir se obtienen los datos del usuario con el correo electrónico proporcionados por el método ***ObtenerUsuario*** (ver indicador 3), posteriormente se comprueba si el correo electrónico de ese usuario ya ha sido confirmado (ver indicador 4), si no ha sido confirmado envía una advertencia (indicador 5) indicando al usuario que tiene que hacer la validación de su correo electrónico, pero en caso de haber validado el correo electrónico, los datos del usuario proporcionados por el método ***ObtenerUsuario*** son almacenados en una clase llamada **Claims** (ver indicador 6) proporcionada por el espacio de nombre **System.Security.Claims**. Al guardar los datos en esta clase (**Claims**) permite crear una identidad del usuario (ver indicador 7) para que sean almacenados en una Cookie, y esto se hace a través del método **SignInAsync** (ver indicador 8) proporcionado por **ASP.NET**

Core, al ejecutarse este método, crea una sesión con los datos del usuario y estos puedan ser consultados (en el Back-End) durante la sesión, facilitando las validaciones al momento que un usuario realice una transacción en la ´plataforma. Y también de esta forma el usuario puede navegar a través de la plataforma de una manera segura.

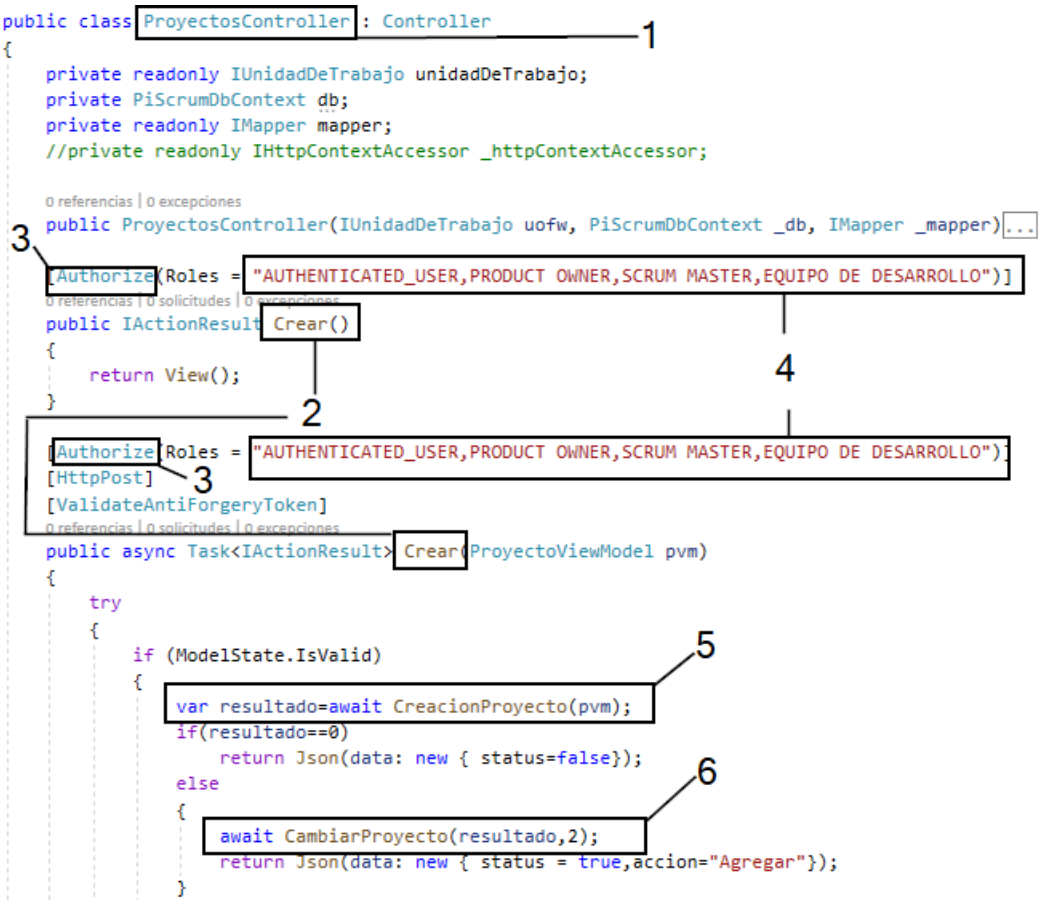


Figura 4.33. Implementación de la acción Crear para la creación de proyectos.

Para poder crear proyectos en la plataforma es necesario registrarse y posteriormente iniciar sesión. Para la creación de proyectos en la plataforma propuesta, se implementó una acción llamada ***Crear*** (ver indicador 2 en la figura 4.33) en un nuevo controlador llamado ***ProyectosController*** (ver indicador 1) el cual contiene las acciones para gestionar los proyectos creados a través de la acción ***Crear***, esta acción responde a los métodos **GET** y **POST**, debido a que el usuario primero solicita una vista para crear un proyecto y posteriormente envía los datos al servidor (a través del

método **POST**) para ser almacenados en la base de datos. En cada uno de los métodos (**GET** y **POST** de la acción ***Crear***) se puede observar un nuevo atributo llamado **Authorize** (ver indicador 3) esto indica que es necesario autenticarse y que solo es autorizado para los usuarios con rol ***AUTHENTICATED_USER, PRODUCT OWNER, SCRUM MASTER, EQUIPO DE DESARROLLO*** (ver indicador 4) y al iniciar sesión el rol por default que obtiene el usuario es ***AUTENTICATED_USER*** para que pueda crear prooyectos .

Cuando el usuario envía los datos introducidos en la vista para la creación de un proyecto llega al método **POST** de la acción ***Crear***, estos datos son almacenado una variable llamada ***pvm*** (ver indicador 5) de tipo ***ProyectoViewModel*** recibida como argumento por el método ***Crear***, después ***pvm*** es enviado como argumento en el método ***CreacionProyecto*** (ver indicador 5), este método contiene el procedimiento la creación y almacenamiento de los datos (en la base de datos) del nuevo proyecto. Si la transacción de la creación de un nuevo proyecto es exitosa, el método ***CreacionProyecto*** retorna el id (variable ***resultado***, ver indicador 5) del nuevo proyecto para que sea pasado como argumento al método ***CambiarProyecto*** (ver indicador 6). El objetivo del método ***CambiarProyecto*** es crear una nueva identidad del usuario con los datos de la identidad actual del mismo (usuario), pero agregando datos (nombre del proyecto, rol del usuario en el proyecto, etc) a la nueva identidad para que posteriormente se destruya la identidad actual y el usuario navegue con una nueva sesión y nueva identidad. De esta forma cuando el usuario seleccione un proyecto tendrá una identidad nueva conforme al rol asignado en el proyecto. Cuando un usuario crea un proyecto el rol que tendrá automáticamente será **product owner**.

En el siguiente apartado se presenta la implementación para agregar colaboradores a los proyectos debido a que la metodología Scrum es colaborativa y cada involucrado en el proyecto tiene un rol.

4.4.3 Back-End: Invitación de colaboradores a los proyectos creados

Para el desarrollo de esta funcionalidad se crearon tres métodos una para obtener los roles (ver figura 4.34), otro para obtener una lista de correos electrónicos (ver figura 4.35)y el ultimo para poder agregar un colaborador al proyecto (ver figura 4.36).

```
[HttpPost]
0 referencias | 0 solicitudes | 0 excepciones
public async Task<JsonResult> ObtenerRoles()
{
    var IdProyecto = Convert.ToUInt64(ConsultarClaim(ClaimTypes.PrimarySid));
    var NumScrumMaster = (await unidadDeTrabajo.DetallesProyectosRepositorio.ObtenerTodos(
        match: x => x.IdProyecto == IdProyecto && x.Rol.NombreRol == "SCRUM MASTER", incl
    var Roles = new List<Roles>();
    if (NumScrumMaster == 0)
        Roles = db.Roles.Where(x => x.NombreRol != "PRODUCT OWNER").ToList();
    else
        Roles = db.Roles.Where(x => x.NombreRol == "EQUIPO DE DESARROLLO").ToList();

    return Json(Roles);   1
}
```

Figura 4.34. Método para obtener los roles en formato JSON.

De la misma manera que es para obtener los roles, para obtener una lista de **correos electrónicos** es retornada a la vista en formato JSON (ver indicador 1 de la figura 4.35). Para obtener una lista de correos electrónicos se realiza una consulta en la tabla ***Usuarios*** (ver indicador 2) de la base de datos ***db_piscrum*** creada en **PostgreSQL**, esto para obtener los **correos electrónicos** que sean diferentes al **correo electrónico** del **usuario** (ver indicador 2) que los está agregando (usuario con rol de **product owner**), para obtener ese correo electrónico se consulta la información almacenada en la Cookie a través del método ***ConsultarClaim*** (ver indicador 3).

```
0 referencias | 0 solicitudes | 0 excepciones
public async Task<JsonResult> ObtenerListaCorreos(string correo)
{
    var CorreoAuth = ConsultarClaim(ClaimTypes.Email);                    3
    var Modelo = (await unidadDeTrabajo.UsuariosRepositorio.ObtenerTodosAsin(
                  match: x => x.EMail != CorreoAuth))
        .Select(x => new { x.EMail }).ToList();
    return Json(Modelo);
}
                                  1                                       2
```

Figura 4.35. Método para obtener una lista de correos electrónicos.

Los métodos presentados anteriormente fueron desarrollados debido a que cuando se requiere agregar un colaborador a un proyecto es necesario de un **correo electrónico** para obtener el

usuario asociado a el, y el **rol** es para establecer que papel tomará el colaborador en el proyecto al cual fue agregado. En la acción **Agregar** de tipo POST (ver indicador 1) que se observa en la figura 4.36 se realizan las acciones mencionadas con anterioridad.

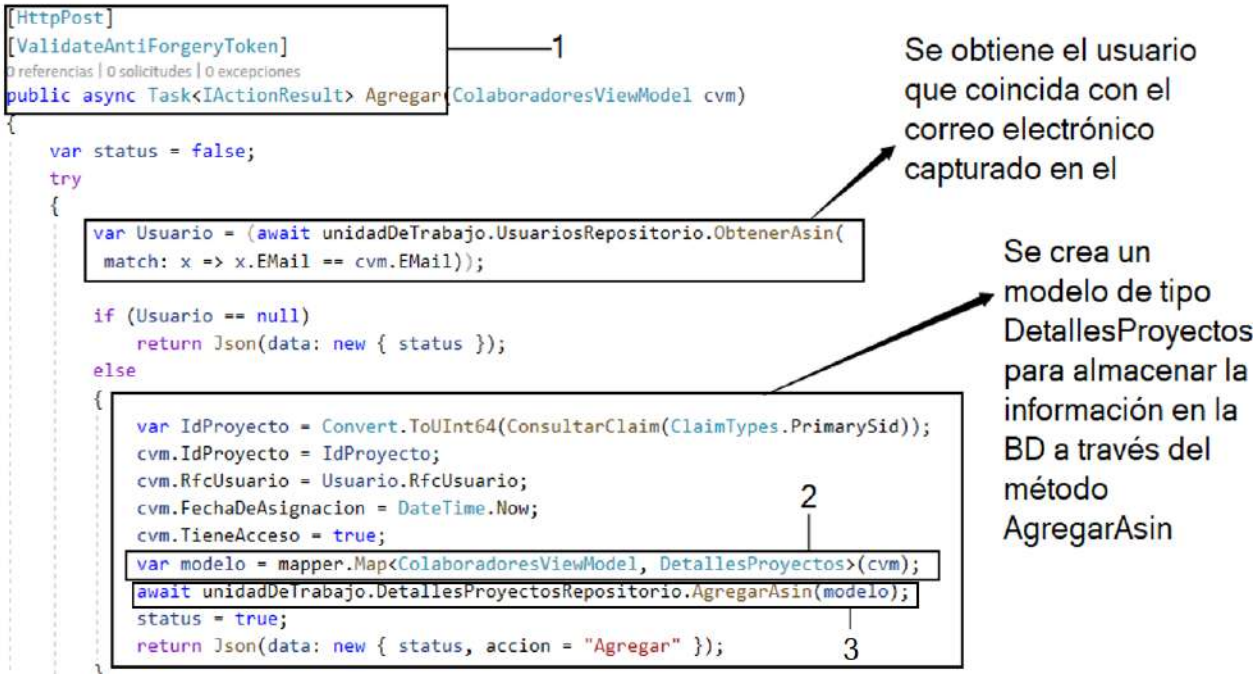


Figura 4.36. Implementación del método para agregar un colaborador a un proyecto.

En las líneas de código de la ilustración 4.37 se muestran las propiedades del ViewModel Colaboradores (ver indicador 1), el cual permite almacenar la información introducida en el formulario por el usuario para posteriormente realizar un mapeo entre **ColaboradoresViewModel** y la clase **DetallesProyectos** (ver indicador 2 en la figura 4.36) esto es debido a que la propiedad (**DetallesProyectosRepositorio**) proporcionada por el patrón **UnidadDeTrabajo** (ver indicador 3 en la figura 4.36) necesita de un objeto (modelo) de tipo **DetallesProyectos** (ver indicador 2 en la figura 4.36) para pasarlo como argumento en el método AgregarAsin (ver indicador 3 en la figura 4.36).

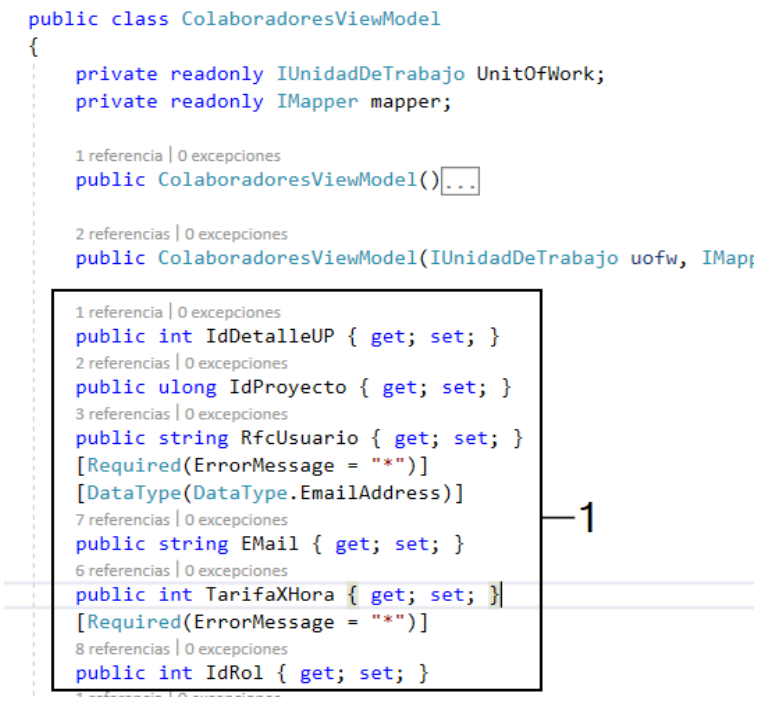


Figura 4.37. Propiedades del ViewModel (Colaboradores).

A continuación, en los siguientes apartados se presentará la implementación del **Front-End** que corresponde al **Back-End** presentado en los apartados anteriores.

4.4.4 Front-End: Formulario para el registro de usuarios en la plataforma

Al principio del Sprint se presento el desarrollo del registro de usuarios del lado del servidor, donde se mostro la acción ***Registrar*** la cual responde a una vista llamada ***Registrar.cshtml***. En la figura 4.38 se puede observar la estructura HTML del archivo ***Registrar.cshtml*** para visualizar un formulario, el cual permite capturar la información para realizar un registro de un nuevo usuario. En la linea 1 se declara el modelo que necesita la vista para poder almacenar la información y este ser enviado a la acción.

Figura 4.38.Vista para registrar un nuevo usuario.

En las lineas de html del la vista ***Registrar.cshtml*** (ver indicador 1) que se muestran al final de la figura 4.39 se agregaron los scripts para hacer uso de los estilos proporcionados por Bootstrap y Jquery para validar los campos del formulario.

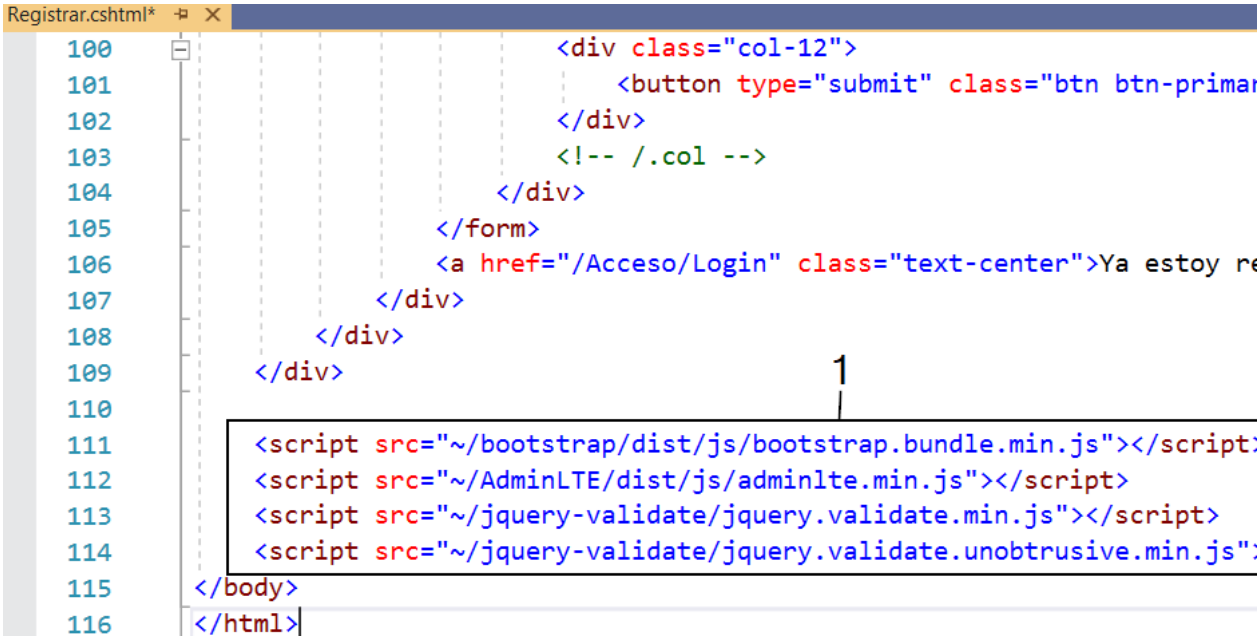


Figura 4.39. Scripts para agregar estilo y mecanismo de validación del formulario registrar.

4.4.5 Front-End: Formulario para iniciar sesión y creación de un nuevo proyecto

En este apartado se presentará el desarrollo de los formularios para iniciar sesión y crear un proyecto. En la figura 4.40 se muestra la estructura HTML del formulario para iniciar sesión con tenido en la vista ***Login.cshtml***, en la etiqueta **<form>** se puede observar una palabra reservada **asp-route-returnurl** (ver indicador 1), la cual es proporcionada por **Razor** que es una sintaxis proporcionada por **C#** que permite utilizarse como motor de programación en las **vistas HTML**. **Asp-route-returnurl** almacena la **url** en caso de querer iniciar sesión en un área restringida y posteriormente re direccionar a la **url** solicitada.

Figura 4.40. Formulario para iniciar sesión en la plataforma.

En las líneas de HTML que se muestran en la figura 4.41 se observa el formulario para crear un nuevo proyecto que está contenido en la vista ***Agregar.cshtml***.

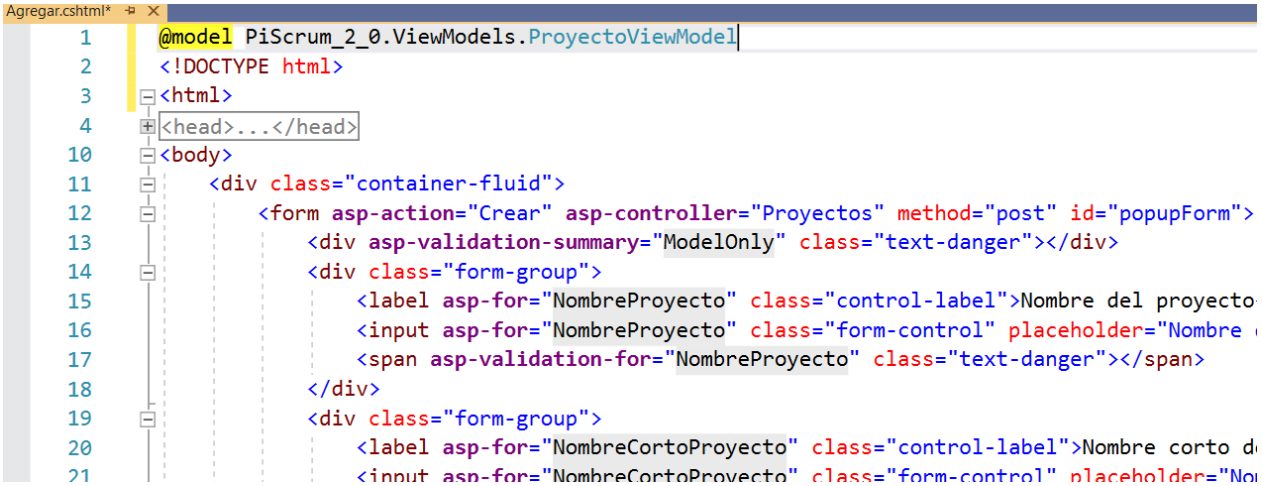


Figura 4.41. Formulario para crear un nuevo proyecto.

En las líneas de código **javascript** que se observan en la figura 4.42 evitan que un usuario al establecer una fecha de finalización en un proyecto no sea mayor a la inicial.

```
<script>
    $(document).ready(function () {
        var settings = {
            validClass: "is-valid",
            errorClass: "is-invalid"

        };
        $.validator.setDefaults(settings);
        $.validator.unobtrusive.options = settings;
    });

    var today = new Date(new Date().getFullYear(), new Date().getMonth(), new Date().getDate());
    $('#FechaInicioEstS').datepicker({
        uiLibrary: 'bootstrap4',
        iconsLibrary: 'fontawesome',
        minDate: today,
        format: 'dd-mm-yyyy',
        modal: true,
        header: true,
        footer: true,
        maxDate: function () {
            return $('#FechaFinEstS').val();
        }
    });
    $('#FechaFinEstS').datepicker({
        uiLibrary: 'bootstrap4',
        iconsLibrary: 'fontawesome',
        format: 'dd-mm-yyyy',
        modal: true,
        header: true,
        footer: true,
        minDate: function () {
            return $('#FechaInicioEstS').val();
```

Figura 4.42. Script para evitar que la fecha de finalización sea menor a la inicial.

En la figura 4.43 se muestra el formulario para agregar un colaborador a un proyecto, donde se necesita introducir un correo electrónico, un rol y numero entero que representa una cantidad monetaria que ganara el colaborador por hora (tarifa por hora) invertidas en las tareas asignadas. El correo electrónico, rol y la tarifa por hora son almacenados en las propiedades ***Email***, ***IdRol*** y ***TarifaXHora*** (ver indicador 1), estas propiedades son proporcionadas por el ViewModel (***ColaboraboradoresViewModel***) que se declara al inicio de la vista ***ConsultaColaboradores.cshtml*** (ver indicador 2). Este ***ViewModel*** es recibido con los datos introducidos por el usuario como parámetro en la acción ***Agregar*** (ver figura 4.36).

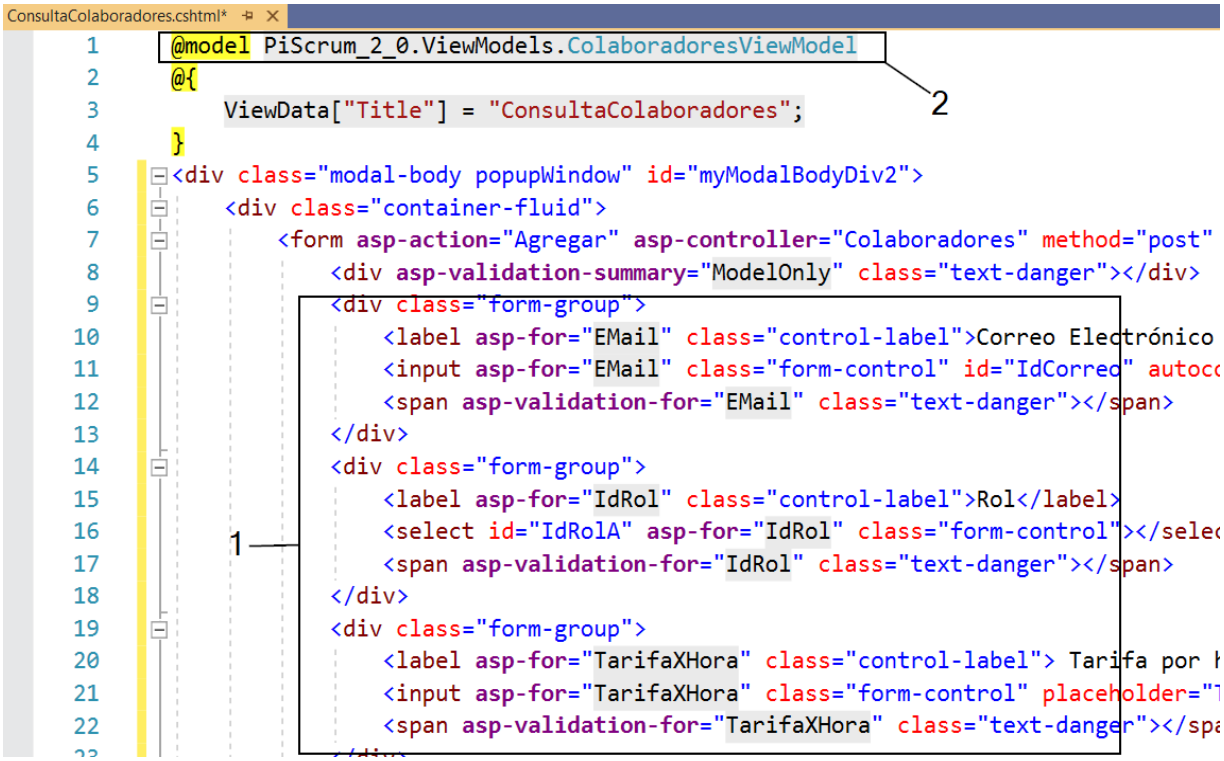


Figura 4.43. Formulario HTML para agregar un colaborador a un proyecto.

De lado del servidor se implementó un método (***ObtenerListaCorreos***) que se encuentra en el controlador Colaboradores, el cual retorna a la vista ***ConsultaColaboradores.cshtml*** (ver figura 4.44) una lista de correos electrónicos en formato JSON.

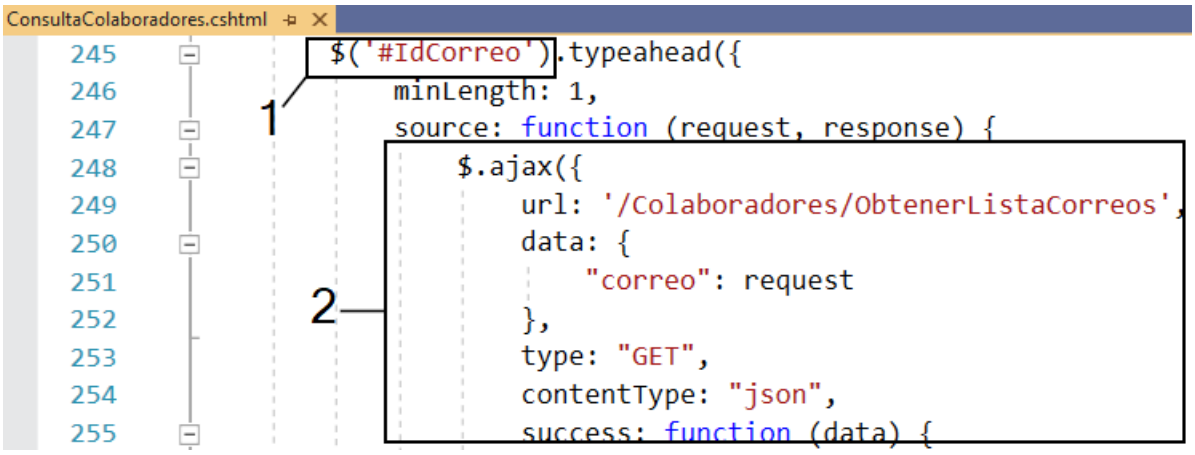


Figura 4.44. Función para solicitar una lista de correos electrónicos a través de AJAX.

Las líneas que código javascript que se muestran en la figura 4.44 se observa que se hace una petición al método ***ObtenerListaCorreos*** a través de AJAX (ver indicador 2) permitiendo obtener la lista sin necesidad de refrescar la vista. Esta lista es proporcionada a la propiedad ***Email***

contenida en el formulario para agregar un colaborador (ver indicador 1 en la figura 4.43), a través del identificador ***IdCorreo*** (ver indicador 1 de la figura 4.44).

En este apartado se presentó el desarrollo de las funciones de lado del servidor para el registro de usuarios en la plataforma para poder acceder a ella, además se presentó el mecanismo para autenticar y autorizar los usuarios registrados, permitiendo la creación de proyectos y por último poder agregar colaboradores a un proyecto. Todas estas funciones fueron creadas con sus respectivas vistas para la captura de información. A continuación, en el tercer sprint se mostrará el desarrollo para registrar las historias de usuario, sprints y priorización de historias de usuario.

4.5 Sprint 3: Gestión de las historias de usuario y creación de un nuevo proyecto

En la tabla 4.7 se presenta la planeación del Sprint 3, donde cada historia de usuario se desglosa en tareas que tienen un tiempo estimado para su realización, así como el inicio y finalización de este Sprint.

Tabla 4.7. Planeación del sprint 3.

Fecha de inicio		**Fecha de finalización**		**N° semanas**	
04-08-2019		19-10-2019		11	
Historia de usuario	**Tareas**	**Fecha de inicio**	**Fecha de finalización**	**Horas de trabajo de la tarea**	**Horas de trabajo total de la historia de usuario**
	Crear una función para registrar las historias de usuario en el product backlog.	04-08-2019	08-08-2019	35	
	Implementar un método para consultar el product backlog.	08-08-2019	11-08-2019	25	

HU 5	Desarrollar una función que permita modificar la información de las historias de usuario del product backlog.	11-08-2019	15-08-2019	32	**156**
	Crear un método que permita al usuario con rol de product owner, eliminar las historias de usuario del product backlog.	15-08-2019	19-08-2019	32	
	Implementar una vista que contenga un formulario para que el usuario pueda registrar las historias de usuario.	19-08-2019	21-08-2019	15	
	Desarrollar una vista para modificar la información de las historias de usuario a través de un formulario.	21-08-2019	23-08-2019	17	
HU 6	Crear una función que permita al usuario con rol de product owner, crear Sprints.	23-08-2019	28-08-2019	39	**170**
	Implementar un método que permita a los usuarios con rol de equipo de desarrollo, Scrum master y product owner, consultar la información de los Sprints.	28-08-2019	31-08-2019	25	
	Desarrollar una función que permita al usuario con rol de product owner, modificar la información de los Sprints.	31-08-2019	04-09-2019	30	

	Implementar un método que permita al usuario con rol de product owner, eliminar los Sprints.	04-09-2019	08-09-2019	36	
	Crear una vista, donde el usuario pueda crear un Sprint.	08-09-2019	10-09-2019	20	
	Desarrollar una vista que permita al usuario modificar la información de un Sprint a través de un formulario	10-09-2019	13-09-2019	20	
HU 7	Desarrollar un método para que el usuario con rol de product owner priorice las historias de usuario en el product backlog y Sprints.	13-09-2019	19-09-2019	50	**80**
	Crear una vista de árbol (treeview) para facilitar al usuario la priorización de las historias de usuario.	19-09-2019	23-09-2019	30	
HU 8	Crear una función que permita consultar información de las historias de usuario únicamente a los usuarios con rol equipo de desarrollo o Scrum master.	23-09-2019	26-09-2019	25	**40**
	Crear una vista que permita visualizar la información de las historias de usuario consultas.	26-09-2019	28-09-2019	15	
HU 9	Implementar un método para validar si el usuario que está	28-09-2019	10-10-2019	100-6	**100**

	moviendo una historia de usuario a un Sprint, tiene el rol de product owner.				
HU 10	Desarrollar una función que retorne el estatus de los Sprints.	10-10-2019	19-10-2019	70	**70**
		Total de horas de trabajo invertidas			**616**

Cada una de estas historias de usuario con sus tareas serán abarcadas en el presente desarrollo de este Sprint, el cual contempla la implementación de la gestión de historias de usuario y creación de Sprints. En este apartado el desarrollo será presentados en dos fases **Back-End** y **Front-End**.

4.5.1 Back-End: Implementación para el registro de historias de usuario en el product backlog y Sprints

```
public class HistoriasUsuarioController : Controller
{
    private readonly IUnidadDeTrabajo unidadDeTrabajo;
    private PiScrumDbContext db;
    private readonly IMapper mapper;

    0 referencias | 0 excepciones
    public HistoriasUsuarioController(IUnidadDeTrabajo uofw, PiScrumDbContext _db, IM
    {
        unidadDeTrabajo = uofw;
        db = _db;
        mapper = _mapper;
    }
    [Authorize(Roles = "PRODUCT OWNER")]
    0 referencias | 0 solicitudes | 0 excepciones
    public IActionResult AgregarHistoria()
    {
        return View();
    }

    [Authorize(Roles = "PRODUCT OWNER")]
    [HttpPost]
    [ValidateAntiForgeryToken]
    0 referencias | 0 solicitudes | 0 excepciones
    public async Task<IActionResult> AgregarHistoria(HistoriasUsuarioViewModel huvm)
    {
```

1

2

Figura 4.45. Acción para agregar una nueva historia de usuario.

La implementación de las acciones (métodos) que se presentan en este apartado, están contenidos en un controlador que fue creado con el nombre de ***HistoriasUsuariosController*** (ver figura 4.45), donde se creó la acción ***AgregarHistoria*** que responde a los métodos **GET** (ver indicador 1) y **POST** (ver indicador 2). El objetivo de esta acción es permitir el registro de las **historias de usuario** en el **product backlog**.

La acción ***AgregarHistoria*** solo responderá a la solicitud de los usuarios con rol de **product owner** (ver indicador 1 y 2), esto es debido a que el **product owner** es quien se encarga de crear un **product backlog** a través de las **historias de usuario** y presentarlo a los demás integrantes. Se implementó dos métodos (**GET** y **POST**), porque primero el usuario con rol de **product owner** solicitará una vista para agregar una nueva historia de usuario al **product backlog**, entonces se ejecutará la acción ***AgregarHistoria*** del método **GET**, posteriormente retornará su vista correspondiente que contiene un formulario que se mostrará su implementación en los apartados relacionados con el **Front-End**. Cuando el usuario envía la información introducida en el formulario de la vista que corresponde a la acción ***AgregarHistoria***, es almacenada en un **ViewModel** llamado ***HistoriasUsuariosViewModel*** y este es recibido en la acción ***AgregarHistoria*** del método **POST** a través del parámetro **huvm** de tipo ***HistoriasUsuarioViewModel*** (ver indicador 2).

La información almacenada en el parámetro ***huvm*** es procesada dentro de la acción ***AgregarHistoria***, para ello primero es necesario asignar un orden o prioridad, donde se realiza una consulta y se contabilizan las historias de usuario registradas a través del método **Count** proporcionado por **LINQ** (ver indicador 1 de la figura 4.46). Posteriormente se almacena la información de la nueva historia de usuario en un objeto de tipo ***HistoriasUsuario*** llamado ***historia***

y es enviado como argumento en el método ***Agregar*** proporcionado por el patrón **repositorio** (ver indicador 2).


```
public async Task<IActionResult> AgregarHistoria(HistoriasUsuarioViewModel huvm)
{
    var status = false;
    try
    {
        ListadoProyectos lp = new ListadoProyectos();
        Random r = new Random();
        var IdProyecto = ConsultarClaim(ClaimTypes.PrimarySid);
        huvm.IdHistoriaUsuario = Convert.ToUInt64(r.Next(1, 100) + DateTime.Now.ToString("ddMMyyy
        var NO = (from hu in await unidadDeTrabajo.HistoriasUsuarioRepositorio.ObtenerTodosAsin(
                  join pb in await unidadDeTrabajo.ProductBacklogRepositorio.ObtenerTodosAsin() o
                  where pb.IdProyecto == Convert.ToUInt64(IdProyecto) && hu.IdSprint == null
                  select hu).Count();

        huvm.NUmOrden = NO + 1;

        huvm.IdProductBacklog = (await unidadDeTrabajo.ProductBacklogRepositorio.ObtenerAsin(
            match: x => x.IdProyecto == Convert.ToUInt64(IdProyecto))).IdProductBacklog;
        var historia = mapper.Map<HistoriasUsuarioViewModel, HistoriasUsuario>(huvm);
        historia.EstimacionEsfuerzo = 0;
        historia.EstimacionEsfuerzoRestante = 0;
        historia.Estatus = "PENDIENTE";
        historia.Resumen = WebUtility.HtmlEncode(huvm.Resumen);
        await unidadDeTrabajo.HistoriasUsuarioRepositorio.AgregarAsin(historia);
```

Figura 4.46. Procedimiento para guardar la historia de usuario en la base de datos.

```
[HttpPost]
[ValidateAntiForgeryToken]
0 referencias | 0 solicitudes | 0 excepciones
public async Task<IActionResult> AgregarSprint(SprintViewModel svm)
{
    try
    {
        //Proceso para crear un sprint
        var IdProyecto = Convert.ToUInt64(ConsultarClaim(ClaimTypes.PrimarySid));
        Random r = new Random();
        svm.IdSprint= Convert.ToUInt64(r.Next(1, 100) + DateTime.Now.ToString("ddMMyyyy") + D
        svm.EstimacionTodalHistorias = 0;
        svm.Estatus ="PENDIENTE";
        svm.IdProductBacklog = (await unidadDeTrabajo.ProductBacklogRepositorio.ObtenerAsin(
            match:x=>x.IdProyecto==IdProyecto)).IdProductBacklog;
        var sp = mapper.Map<SprintViewModel, Sprint>(svm);
        await unidadDeTrabajo.SprintRepositorio.AgregarAsin(sp);
```

Figura 4.47. Acción para agregar un Sprint.

Cuando el usuario con rol de **product owner** haya agregado las historias de usuario al **product backlog**, necesitará presentar un conjunto de historias de usuario con mayor prioridad (**Sprint**

backlog) que serán atendidas en un **Sprint** determinado. Con lo expuesto, es necesario crear un Sprint cada vez que se realice el proceso mencionado, en las líneas de código que se muestran en la figura 4.47 permiten crear **Sprints** en la plataforma.

Después de haber creado un **Sprint** para indicar que historias de usuario se realizarán en el Sprint, se desarrolló un método llamado ***CmbiarNodo*** (ver indicador 1) que contiene las siguientes líneas de código que se muestran en la figura 4.48. Este método actualiza la información de las historias de usuario (asignando un **id** que corresponde a un Sprint al campo ***IdSprint*** de un registro de la tabla ***HistoriasUsuario***, consultar la clase ***HistoriasUsuario*** del diagrama de clases presentado en el capítulo 3) (ver indicador 2) que están siendo agregadas a un **Sprint** o en caso de querer regresar alguna historia de usuario al **product backlog**, entonces el método establecerá el valor de ***IdSprint*** en nulo (ver indicador 3).

```
[HttpPost]
0 referencias | 0 solicitudes | 0 excepciones
public async Task<IActionResult> CambiarPosicionNodo(int id, int parentId, int orderNumber)
{
        foreach (var item in listaActualizada)
        {
            var part = await unidadDeTrabajo.ListadoProyectosRepositorio.ObtenerAsin(
                match:x=>x.ID==item.ParentID);
            foreach (var hist in his)
            {
                if (part.Tipo == "PB" && hist.IdHistoriaUsuario == item.IDTipo)
                {
                    hist.IdSprint = null;
                    hist.NumOrden = item.Orden;
                    listaModificada.Add(hist);
                    break;
                }
                else if (part.Tipo == "ST" && hist.IdHistoriaUsuario == item.IDTipo)
                {
                    hist.IdSprint = part.IDTipo;
                    hist.NumOrden = item.Orden;
                    listaModificada.Add(hist);
                }
            }
        }
        foreach (var item in listaModificada)
        {
            try
            {
                await unidadDeTrabajo.HistoriasUsuarioRepositorio.ActualizarLAsin(item, item.IdHistoriaUsuario);
```

1
2
3

Figura 4.48. Código para mover historias de usuario a un Sprint o product backlog.

4.5.2 Back-End: Implementación para consultar información de las historias de usuario y Sprints

Para la consulta de información de **historias de usuario**, **Sprints** y **tareas** se implementó un solo método (**ConsultarNodo**) que retorna la información en formato JSON (ver indicador 1 en la figura 4.49), esto se hizo debido a que del lado del cliente se implementó una vista de árbol (treeview) para visualizar las **historias de usuarios**, **tareas** y **Sprints**, y para implementar dicha función visual se necesitó crear una clase llamada ***ListadoProyectos*** como se mostró en el diagrama de clases (ver figura 3.8) presentado en el capítulo 3. Esto permite almacenar información de las **historias de usuario**, **Sprints** y **tareas** en una misma estructura, pero solo almacenando datos muy específicos (**nombre**, **tipo**, **idtipo** y **orden**) aunque estos datos son también almacenados en sus clases y tablas (**HistoriasUsuario**, **Sprints**, **Tareas**) correspondientes.

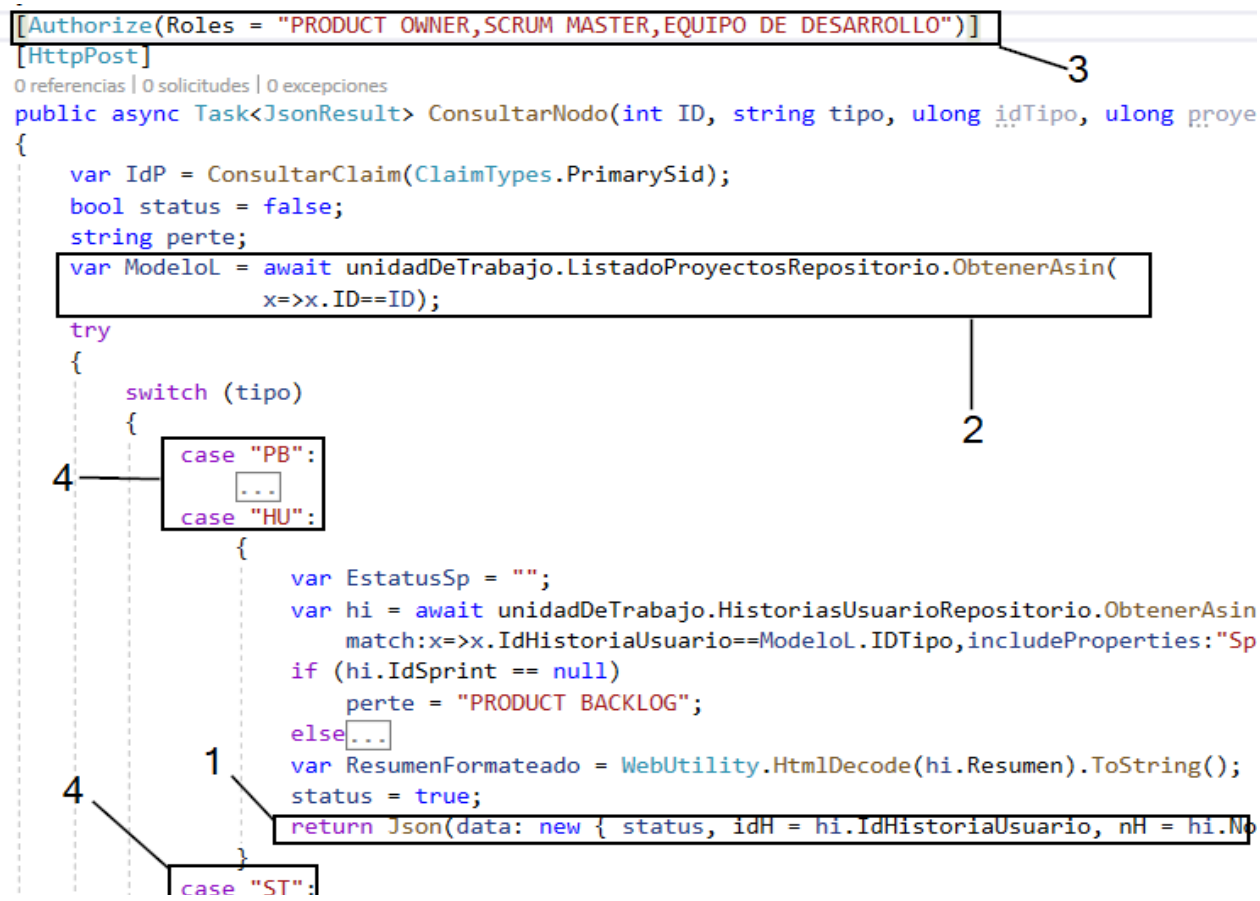


Figura 4.49. Método para consultar la información de una historia de usuario, sprint o tarea.

En las líneas de código de la figura 4.49, se realiza una consulta a la entidad **ListadoProyectos** (ver indicador 2) para obtener la información de la historia de usuario, sprint o tarea solicitada por el usuario y es retornada a una vista.

Cabe mencionar que el método **ConsultarNodo** es autorizado para los usuarios con rol de **product owner**, **Scrum master** o **equipo de desarrollo** (ver indicador 3). La información que retorna el método, depende de lo que solicite el usuario y esto se filtra a través de los casos (ver indicador 4) proporcionados por la instrucción **Switch** proporcionada por **C#**.

En el siguiente apartado se presenta la implementación del lado del cliente para capturar la información de las **historias de usuario** y **Sprints** para su posterior almacenamiento.

4.5.3 Front-End: Formulario para agregar historias de usuario y Sprints

Una vista requiere de un modelo para representar información con una estructura especifica al usuario o para almacenar información proporcionada por el usuario y ser enviada al servidor para ser procesada en una acción de un controlador. Anteriormente se presentó la implementación de las acciones en un controlador para registrar **historias de usuario** y **Sprints** de lado del servidor, en este apartado se presenta la implementación de las vistas que proporcionan el mecanismo para que los usuarios puedan introducir la información de las **historias de usuario** y **Sprints**.

En la figura 4.50 se observa la estructura HTML para la creación de un formulario en la vista ***AgregarHistoria.cshtml***, que permitirá a los usuarios introducir información de las historias de usuario (**Nombre**, **Resumen** y **Descripción**). Los datos proporcionados por el usuario son almacenados en las propiedades (ver indicador 3) proporcionadas por el modelo que es declarado al inicio de la vista (ver indicador 1), esta información contenida en el modelo **ViewModel** ***HistoriasUsuarioViewModel*** es enviada a la acción ***AgregarHistoria*** contenida en el controlador ***HistoriasUsuario*** a través del método **POST** (ver indicador 2).

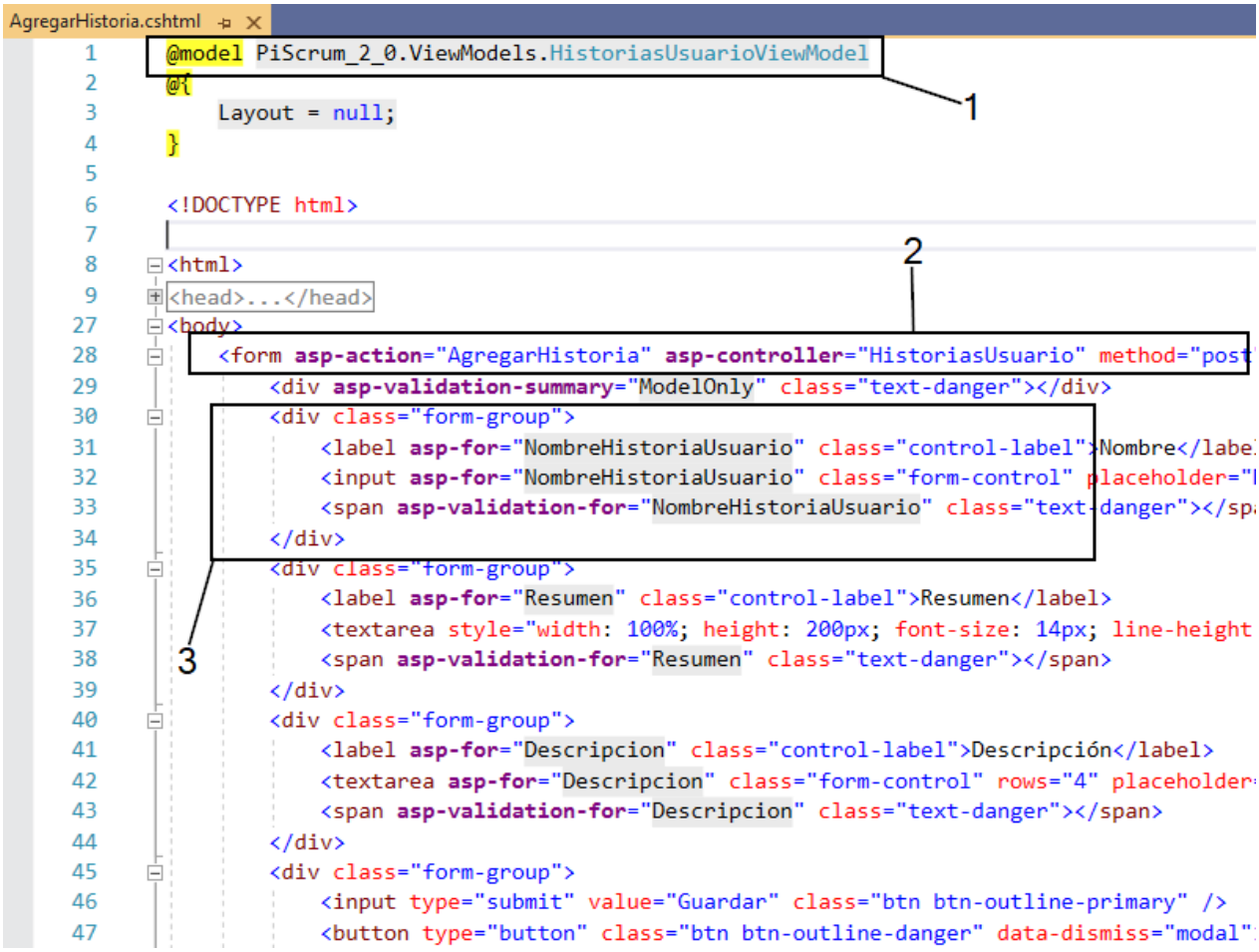


Figura 4.50. Formulario HTML para agregar una historia de usuario.

En la figura 4.51 se muestra la vista que contiene el formulario para crear un **Sprint**. De la misma forma que la anterior vista se declara un modelo (ver indicador 1) que proporciona las propiedades (ver indicador 3) para almacenar la información proporcionada por el usuario y es enviada a la acción ***AgregarSprint*** contenida en el controlador ***Sprints*** a través del método **POST** (ver indicador 2).

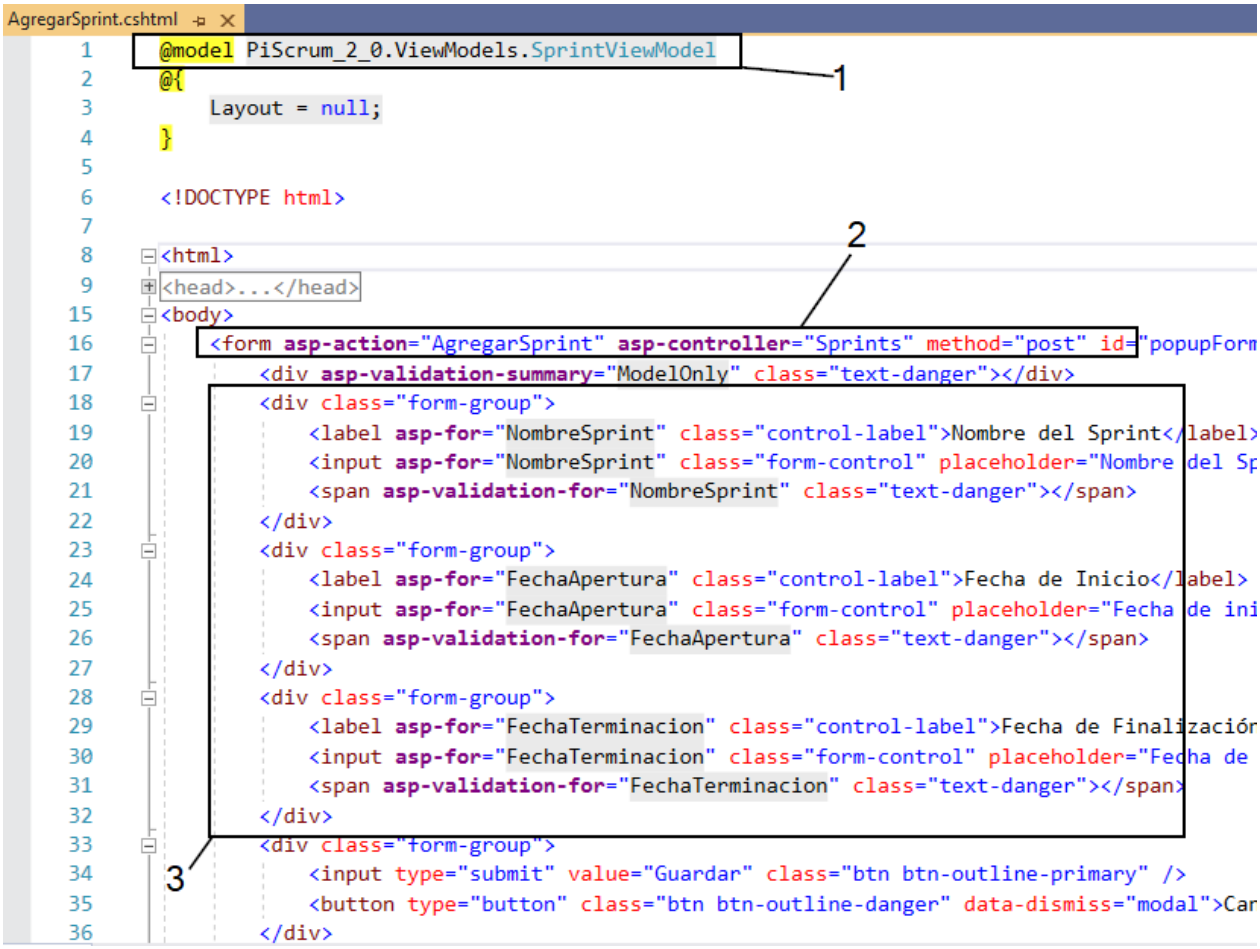


Figura 4.51. Formulario HTML para crear un Sprint.

4.5.4 Front-End: Implementación del treeview y el mecanismo para priorizar las historias de usuario

Las siguientes lineas de codigo javascript (ver figura 4.52) permiten generar el **treeview** para visualizar las **historias de usuario**, **Sprints** y **tareas** en forma de arbol, el codigo javascript carga las **historias de usuario**, **Sprints** y **tareas** en la etiqueta **DIV** a traves del atributo **id** (ver línea 251) con la referencia ***tree*** (ver lineas 251 y 253). La informacion que es visualizada en el **DIV** es almacenada en el parametro dataSource (línea 260) que es proporcionada por la accion ***ObtenerHistorias*** contenida en el controlador ***HistoriasUsuario***, la cual retorna la informacion en formato **JSON**, para que los usuario puedan priorizar las historias de usuario simplemente moviendolas a la posición deseada , en el parametro **dragAndDrop** se establecio el valor de true,

permientiendo habilitar la funcion de soltar y arrastrar objetos del **DOM**, esta funcion es proporcionada por **JQuery**.

```
251  <div class="col-lg-5 scrollchy treeviewH" id="tree">
252  </div>
253  @section Scripts{
254      <script type="text/javascript">
255
256      var tree;
257
258      tree = $('#tree').tree({
259          primaryKey: 'id',
260          dataSource: '/HistoriasUsuario/ObtenerHistorias',
261          dragAndDrop: true,
262          imageCssClassField: 'icon',
263          uiLibrary: 'bootstrap4'
264      });
```

Sección HTML para mostrar el treeview

Código javascript para generar el

Figura 4.52. Implementación de la función para generar el treeview.

Después de cargar la información en forma de árbol con la implementación anterior y la función de soltar y arrastrar para priorizar las historias de usuario, es necesario almacenar las posiciones de las historias de usuario cuando sean cambiadas, para ello se implementó el siguiente código javascript que se muestra en la figura 4.53. Cuando se cambia la posición de un nodo (historia de usuario) se obtiene la información (id del padre destino, numero de orden y su id) de este a través del evento **nodeDrop** (línea 48), entonces la información es enviada al servidor a través de **AJAX** con el método **POST** (línea 53) al método ***CambiarPosicionNodo*** contenido en el controlador ***HistoriasUsuario*** (línea 51).

```
47      //método para cambiar el nodo
48      tree.on('nodeDrop', function (e, id, parentId, orderNumber) {
49          var params = { id: id, parentId: parentId, orderNumber: orderNumber };
50          $.ajax({
51              url: '/HistoriasUsuario/CambiarPosicionNodo',
52              data: params,
53              type: 'POST',
54              success: function (data)...
102         })
103             .fail(function () {
104                 alert('error.');
105             });
106     });
```

Figura 4.53. Función para ordenar las historias de usuario.

4.5.5 Front-End: Implementación de la función para mostrar la información de las historias de usuario y Sprints

Cuando se muestra los nodos (historias de usuario) en el **treeview** con método **on** implementando el evento **select** (línea 109) proporcionado por **JQuery** es posible obtener la información del nodo seleccionado a través de **AJAX** el cual hace una petición a la acción ***ConsultarNodo*** del controlador ***HistoriasUsuario*** (línea 113) y este retorna la información en formato **JSON** (ver figura 4.54).

```
109     tree.on('select', function (e, node, id, ) {
110         var data = tree.getDataById(id);
111         var parametros = { ID: data.id, tipo: data.tipo, idTipo: da
112         $.ajax({
113             url: '/HistoriasUsuario/ConsultarNodo',
114             data: parametros,
115             type: 'POST',
116             success: function (data) {
117                 if (data.status) {
118                     if (data.tipoA === "PB")...
134                     if (data.tipoA === "ST")...
167                     if (data.tipoA === "HU")...
208                     if (data.tipoA === "TA")...
```

Figura 4.54. Función para obtener la información de las historias de usuario.

4.6 Sprint 4: División de las historias de usuario en tareas y creación de tipos de trabajos

En la tabla 4.8 se presenta la planeación del Sprint 4, donde cada historia de usuario se desglosa en tareas que tienen un tiempo estimado para su realización, así como el inicio y finalización de este Sprint.

Tabla 4.8. Planeación del Sprint 4.

Fecha de inicio		**Fecha de finalización**		**N° semanas**	
20-10-2019		11-11-2019		3	
Historia de usuario	**Tarea**	**Fecha de inicio**	**Fecha de finalización**	**Horas de trabajo de la tarea**	**Horas de trabajo total de la historia de usuario**
HU 11	Crear una función que permita asignar la estimación de esfuerzo (tiempo) a una historia de usuario con base a las tareas agregadas.	20-10-2019	21-10-2019	9	**24**
	Crear una función que permita restar la estimación de esfuerzo (tiempo) a una historia de usuario, si una tarea es eliminada.	21-10-2019	22-10-2019	15	
HU 12	Implementar una función que permita únicamente al usuario con rol de equipo de desarrollo crear tareas en las historias de usuario.	23-10-2019	26-10-2019	27	**40**

	Desarrollar una vista que proporcione un formulario para agregar tareas a las historias de usuario.	26-10-2019	27-10-2019	13	
HU 13	Crear un método para modificar la información de las tareas y que solamente sea accesible a los usuarios con rol de equipo de desarrollo.	28-10-19	29-10-19	16	**32**
	Implementar una vista que permita visualizar y modificar la información de una tarea, a través de un formulario.	30-10-19	30-10-19	4	
	Desarrollar una función que permita a los usuarios con rol de equipo de desarrollo eliminar tareas de una historia de usuario.	30-10-19	31-10-19	12	
HU 14	Crear un método que no permita agregar, modificar o eliminar una tarea de la historia de usuario, el estado del Sprint está en proceso o finalizado.	01-11-19	03-11-19	19	**24**
	Implementar una función en el Front-End que oculte o muestre las opciones de agregar, modificar o	03-11-19	03-11-19	5	

	eliminar una tarea con base al estado del Sprint.				
HU 15	Desarrollar un método que permita a los usuarios con rol de equipo de desarrollo asignarse tareas.	04-11-19	05-11-19	12	**24**
	Implementar una función que permita a los usuarios con rol de equipo de desarrollo desasignarse tareas.	05-11-19	06-11-19	7	
	Crear un método en el Front-End que permita mostrar u ocultar botones para asignar o desasignar tareas.	06-11-19	06-11-19	5	
HU 16	Implementar una función que permita crear tipos de trabajos que puedan ser aplicados en las tareas.	07-11-19	07-11-19	5	**33**
	Crear una vista que proporcione un formulario para introducir la información de un nuevo tipo de trabajo.	07-11-19	07-11-19	3	
	Crear un método que permita devolver una lista de tipos de trabajos y pueda ser filtrada con base al nombre del tipo de trabajo.	08-11-19	08-11-19	6	

	Implementar una vista para visualizar la lista de tipos de trabajos que tiene un proyecto.	08-11-19	09-11-19	5	
	Desarrollo una función que permita a los usuarios con rol de equipo de desarrollo, modificar la información de los tipos de trabajos.	09-11-19	09-11-19	5	
	Crear una vista que proporcione un formulario que visualice y permita modificar la información de los tipos de trabajos.	10-11-19	10-11-19	3	
	Implementar una función que permita eliminar tipos de trabajos en un proyecto.	10-11-19	11-11-19	6	
Total de horas de trabajo invertidas					**177**

En los siguientes apartados se presenta el desarrollo de las historias de usuario y tareas que se presentaron en la planeación de este Sprint, la implementación que abarca este Sprint está dividida en Back-End y Front-End.

4.6.1 Back-End: Implementación de la función para la creación de tipos de trabajos

En este apartado se presenta el desarrollo de una función para la creación de tipos de trabajos que podrán ser asignados cuando un usuario con rol de equipo de desarrollo cree tareas. Esta función solo puede ser autorizada para usuarios con rol de equipo de desarrollo debido a que ellos son quienes realizan el proceso de la división de tareas, asignación de tiempo estimado y establecimiento del tipo de trabajo se realizará en cada una de las tareas.

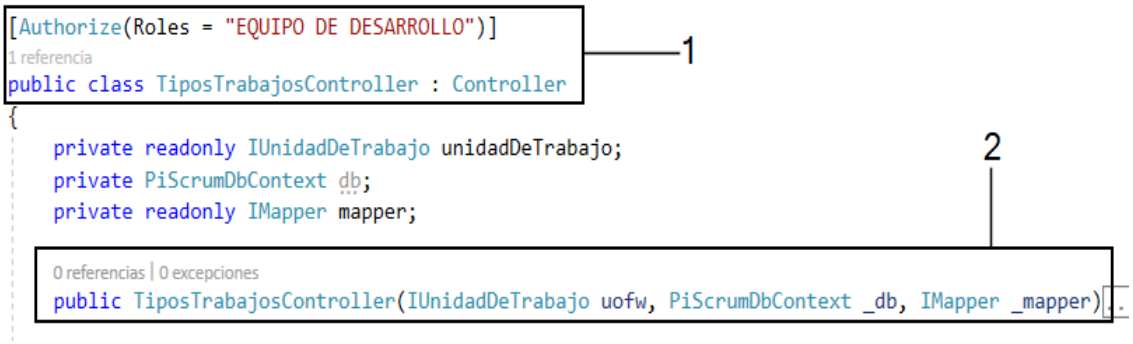


Figura 4.55. Controlador de tipos de trabajos.

Para gestionar los tipos de trabajos se creó un controlador llamado ***TiposTrabajos*** que contendrá toda la lógica de negocio para la gestión de los tipos de trabajos. Como se mencionó, este proceso solamente lo realiza el equipo de desarrollo y por tal motivo el controlador ***TiposTrabajos*** (ver figura 4.55) es solamente accesible para usuarios con rol de equipo de desarrollo (ver indicador 1). Este controlador también requiere de los servicios del contexto (***PiScrumDbContext***), del patrón unidad de trabajo (***IUnidadDeTrabajo***) y de **AutoMapper** (ver indicador 2).

Figura 4.56. Acción para agregar un tipo de trabajo a un proyecto.

Como se han mostrado en las anteriores implementaciones de las acciones, para agregar un tipo de trabajo se creó de una acción llamada ***Agregar*** que responde a un método GET y POST (ver figura 4.56).

En el método **GET** simplemente retorna una vista sin enviar un modelo debido a que cuando se requiera crear un tipo de trabajo no se requiere de datos previos para dicha acción. La acción ***Agregar*** en el método **POST** recibe como parámetro un ViewModel de ***TipoTrabajoViewModel*** (ver indicador 1), el cual es procesado dentro de la acción donde se realiza un mapeo (línea 100) entre las clases ***TipoTrabajoViewModel*** y ***TipoTrabajo*** para obtener un objeto de ***TipoTrabajo*** con datos provenientes del ViewModel recibido por la acción. Con el objeto ***TipoTrabajo*** se agrega un registro a la tabla ***TipoTrabajo*** (línea 101), posterior a esto se realiza otro mapeo entre las clases ***TipoTrabajoViewModel*** y ***DetalleProyectoTiposTrabajos*** para obtener un objeto (línea 102) y este sea como argumento para el método ***AgregarAsin*** para realizar un registro en la tabla ***DetalleProyectoTiposTrabajos*** (línea 103). Con lo anterior, cada proyecto tendrá su lista de trabajos para las tareas a realizar. Esta acción no retorna una vista sino un **JSON** (línea 104) debido a que los datos son enviados a través de **AJAX** y este necesita de una repuesta en formato **JSON**. Con la implementación de esta función es posible realizar el proceso para crear una tarea.

4.6.2 Back-End: Implementación de las funciones para la creación de las tareas en las historias de usuario

```
35      public async Task<IActionResult> Agregar(ulong IdHistoriaUsuario)
36      {
37          TipoTrabajoViewModel ttvm = new TipoTrabajoViewModel(unidadDeTrabajo,mapper);
38          var IdProyecto = Convert.ToUInt64(ConsultarClaim(ClaimTypes.PrimarySid));
39          var datos= await ttvm.ObtenerTrabajos(IdProyecto);
40          ViewBag.TipoTrabajo = new SelectList(datos, "NombreTrabajo", "NombreTrabajo");
41          ViewBag.IdHistoriaUsuario = IdHistoriaUsuario;
42          return View();
43      }
44
45      [HttpPost]
46      [ValidateAntiForgeryToken]
        0 referencias | 0 solicitudes | 0 excepciones
47      public async Task<IActionResult> Agregar(TareasViewModel tvm)
48      {
```

1

Figura 4.57. Acción para agregar tareas a las historias de usuario.

Se implementó una acción llamada ***Agregar*** para agregar una tarea a una historia de usuario que responde a los métodos **GET** y **POST** (ver figura 4.57), en el método **GET** la acción Agregar

recibe como parámetro un **id** de una **historia de usuario** (ver indicador 1) esto para que sea enviado al formulario de registro a través de un **ViewBag** (líneas 41 y 42), el cual es un mecanismo proporcionado por ASP.NET Core para enviar datos a la vista sin necesidad de un modelo.

En el método **GET** se invoca un método llamado ***ObtenerTrabajos*** (línea 39), el cual recibe como argumento el **id** del **proyecto** que es proporcionado a través de la consulta de los **claims** que almacenan información de la **sesión** (línea 38), el método ***ObtenerTrabajos*** proporciona una **lista de trabajos** donde en la línea 40 se envía esta lista de trabajos a través de un **ViewBag** para que en el formulario de registro de tareas se muestre un *combo box* (dropdown list) de los tipos de trabajos y el usuario con facilidad pueda seleccionar uno.

El método **POST** de la acción ***Agregar*** recibe un ViewModel llamado ***TareasViewModel*** (ver indicador 1 de la figura 4.58), el cual almacena la información introducida por el usuario, a través del formulario que se visualiza en la vista.

```
45      [HttpPost]
46      [ValidateAntiForgeryToken]
        0 referencias | 0 solicitudes | 0 excepciones
47      public async Task<IActionResult> Agregar(TareasViewModel tvm)
48      {
49          var status = false;
50          try
51          {
52              //obtener el estado del sprint y historia de usuario
53              var modeloH = await unidadDeTrabajo.HistoriasUsuarioRepositorio.ObtenerAsin(
54                            x => x.IdHistoriaUsuario == tvm.IdHistoriaUsuario);
55              var modeloS = await unidadDeTrabajo.SprintRepositorio.ObtenerAsin(
56                            x=>x.IdSprint==modelo
57              if(modeloH.Estatus=="PROCESO" && mo
58                  return Json(data: new { status, accion = "Agregar_T" });
59              //registro de la tarea
60              var IdProyecto = Convert.ToUInt64(ConsultarClaim(ClaimTypes.PrimarySid));
61              Random r = new Random();
62              tvm.IdTarea = Convert.ToUInt64(r.Next(1, 100) + DateTime.Now.ToString("ddMMyyyy") + Dat
63              tvm.EstimacionEsfuerzoRestante = tvm.EstimacionEsfuerzo;
64              tvm.Estatus = "PENDIENTE";
65              tvm.NumOrden = 1 + (await unidadDeTrabajo.TareasRepositorio.ObtenerTodosAsin(
66                  match:x=>x.IdHistoriaUsuario==tvm.IdHistoriaUsuario)).Count();
67              var modeloT = mapper.Map<TareasViewModel,Tareas>(tvm);
68              await unidadDeTrabajo.TareasRepositorio.AgregarAsin(modeloT);
```

(Campo) readonly IUnidadDeTrabajo TareasController.unidadDeTrabajo

1

2

Figura 4.58. Acción Agregar en el método POST para registrar una tarea.

Este ViewModel es procesado para que se asigné un **id**, **estimación de esfuerzo restante**, **estatus** y un **orden** (ver indicador 2) a una tarea, en la línea 67 se realiza un mapeo entre las clases ***TareasViewModel*** y ***Tareas*** para obtener un objeto llamado ***modeloT*** que tiene los datos proporcionados por el objeto del ViewModel. El objeto ***modeloT*** es proporcionado como argumento al método ***AgregarAsin*** (línea 68) donde agregará un nuevo registro a la tabla ***Tareas*** a través de la entidad ***Tareas***.

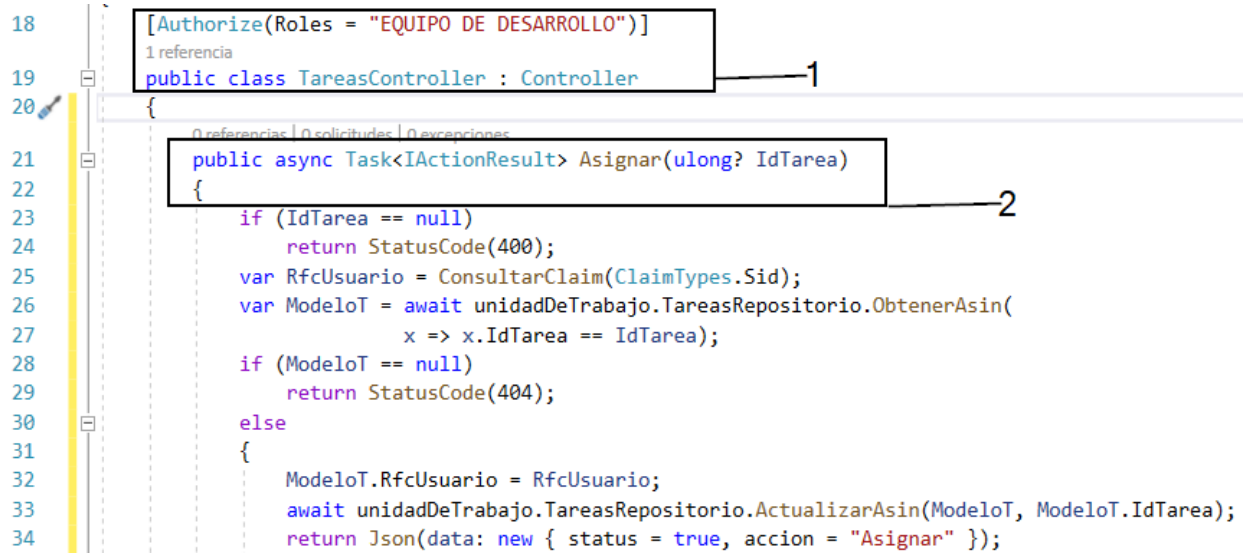
Posteriormente al registro de una **tarea**, se realiza un incremento en la **estimación de esfuerzo de la historia de usuario** (líneas 84 y 85), a la cual se le agrega una nueva **tarea** (ver figura 4.59). Cuando se realiza el incremento se actualiza la información de la historia de usuario a través del método ***ActualizarAsin*** (línea 86).

```
83              //actualizar el esfuerzo estimado de la historia de usuario
84              modeloH.EstimacionEsfuerzo += Convert.ToInt32(modeloT.EstimacionEsfuerzo);
85              modeloH.EstimacionEsfuerzoRestante += Convert.ToInt32(modeloT.EstimacionEsfuerzo);
86              await unidadDeTrabajo.HistoriasUsuarioRepositorio.ActualizarAsin(modeloH,modeloH.IdHistor
87              status = true;
88              return Json(data: new { status,accion="Agregar_T"});
```

Figura 4.59. Código para realizar un incremento en la estimación de esfuerzo de la historia de usuario.

4.6.3 Back-End: Implementación de la función para la asignación de tareas

Para la asignación de una tarea a un integrante del equipo de desarrollo se implementó una acción llamada ***Asignar*** (ver indicador 2 de la figura 4.60) contenida en el controlador ***Tareas*** (ver indicador 1). Esta acción solo responde al método GET y es solo accedido por los usuarios con rol de equipo de desarrollo (ver indicador 1), debido a que en la metodología Scrum los integrantes del equipo de desarrollo son auto organizados y son quienes eligen que tareas realizarán.




```
18      [Authorize(Roles = "EQUIPO DE DESARROLLO")]
        1 referencia
19      public class TareasController : Controller
20      {
            0 referencias | 0 solicitudes | 0 excepciones
21          public async Task<IActionResult> Asignar(ulong? IdTarea)
22          {
23              if (IdTarea == null)
24                  return StatusCode(400);
25              var RfcUsuario = ConsultarClaim(ClaimTypes.Sid);
26              var ModeloT = await unidadDeTrabajo.TareasRepositorio.ObtenerAsin(
27                          x => x.IdTarea == IdTarea);
28              if (ModeloT == null)
29                  return StatusCode(404);
30              else
31              {
32                  ModeloT.RfcUsuario = RfcUsuario;
33                  await unidadDeTrabajo.TareasRepositorio.ActualizarAsin(ModeloT, ModeloT.IdTarea);
34                  return Json(data: new { status = true, accion = "Asignar" });
```

Figura 4.60. Acción para asignar una tarea a los integrantes del equipo de desarrollo.

La acción ***Asignar*** recibe como parámetro un **id** de la **tarea** (ver indicador 2) que será asignada a un **integrante del equipo de desarrollo**, para ello se consulta el **rfc** del usuario (línea 25) contenido en los **claims** de la **sesión**, posteriormente se realiza una consulta a la tabla Tareas con el método ***ObtenerAsin*** (línea 183) que retorna un modelo con datos de tipo ***Tareas*** y es almacenado en la variable ***ModeloT***. En la línea 32 se asigna el rfc (obtenido por la consulta hecha a los **claims**) a la propiedad **RfcUsuario** del modelo ***Tareas*** (***ModeloT***), con la actualización de la propiedad **RfcUsuario** el modelo ***Tareas*** y su **id** son proporcionados como argumentos para el método **AcutualizarAsin** (línea 33) para realizar los cambios en la base de datos y que la tarea sea asignada al usuario con el rfc correspondiente.

4.6.4 Front-End: Implementación del formulario para agregar un nuevo tipo de trabajo

La figura 4.61 se observa la maquetación de la vista ***Agregar.cshtml*** para visualizar el formulario que permite la creación de tipos de trabajos, en la etiqueta **<form>** se puede observar que los datos son enviados a través del método **POST** a la acción ***Agregar*** contenida en el controlador ***TiposTrabajos*** (línea 15). Al realizar el **submit** los datos introducidos por el usuario

son almacenados en el ViewModel **TipoTrabajoViewModel** declarado en la línea 1 de la vista como modelo y posteriormente enviados al servidor.

Agregar.cshtml

```
1   @model PiScrum_2_0.ViewModels.TipoTrabajoViewModel
2   @{
3       Layout = null;
4   }
5
6   <!DOCTYPE html>
7
8   <html>
9   <head>
10      <meta name="viewport" content="width=device-width" />
11      <title>Crear</title>
12  </head>
13  <body>
14      <div class="container-fluid">
15          <form asp-action="Agregar" asp-controller="TiposTrabajos" method="post"
16              <div asp-validation-summary="ModelOnly" class="text-danger"></div>
17              <div class="form-group">
18                  <label asp-for="NombreTrabajo" class="control-label">Nombre del
19                  <input asp-for="NombreTrabajo" class="form-control" placeholder
20                  <span asp-validation-for="NombreTrabajo" class="text-danger"></
21              </div>
22              <div class="form-group">
23                  <input type="submit" value="Guardar" class="btn btn-success" />
```

Figura 4.61. Formulario HTML para agregar un nuevo tipo de trabajo.

4.6.5 Front-End: Implementación del formulario para la creación de tareas

En la implementación de la acción (***Agregar***) para agregar una tarea, en el método **GET** se obtuvo una **lista de trabajos** y se almacenó en un **ViewBag** para poder enviar esa lista a la vista. En la figura 4.62 se puede observar la maquetación de la vista ***Agregar.cshtml***, en la cual se construyó un formulario para que el usuario con rol de **equipo de desarrollo** capture los datos necesarios para agregar una nueva tarea. En la línea 43 se maqueta el combo box (etiqueta HTML **select**) que recibe la lista proporcionada por el ***ViewBag.TipoTrabajo*** implementado en el servidor, además en la línea 23 se puede observar un input (control HTML de entrada) oculto donde se establece un valor proporcionado por el ***ViewBag.IdHistoriaUsuario*** enviado desde el servidor. De lado del servidor la acción ***Agregar*** recibe un ViewModel llamado **TareasViewModel** que es

enviado desde la vista (línea 1) a través del método **POST** como se observa en la línea 21 (ver figura 4.62).

Agregar.cshtml*

```
1   @model PiScrum_2_0.ViewModels.TareasViewModel
2   @{Layout = null;}
3   <!DOCTYPE html>
4   <html>
5   <head>...</head>
20  <body>
21      <form asp-action="Agregar" asp-controller="Tareas" method="post" id="popupForm">
22          <div asp-validation-summary="ModelOnly" class="text-danger"></div>
23          <input asp-for="IdHistoriaUsuario" type="hidden" value="@ViewBag.IdHistoriaUsuario"/>
24          <div class="form-group">
25              <label asp-for="NombreTarea" class="control-label">Nombre</label>
26              <input asp-for="NombreTarea" class="form-control" placeholder="Nombre de la tarea" />
27              <span asp-validation-for="NombreTarea" class="text-danger"></span>
28          </div>
29          <div class="form-group">
30              <label asp-for="Descripcion" class="control-label">Descripción</label>
31              <textarea asp-for="Descripcion" class="form-control" rows="4" placeholder="Descripción de l
32              <span asp-validation-for="Descripcion" class="text-danger"></span>
33          </div>
34          <div class="form-group">
35              <label asp-for="EstimacionEsfuerzo" class="control-label">Estimación de esfuerzo (hrs)</lab
36              <input asp-for="EstimacionEsfuerzo" class="form-control" placeholder="Estimación de esfuerz
37              <span asp-validation-for="EstimacionEsfuerzo" class="text-danger"></span>
38          </div>
39          <div class="form-group">
40              <label asp-for="TipoTRabajo" class="control-label">Trabajo a realizar</label>
41              <select asp-for="TipoTRabajo" class="form-control" asp-items="ViewBag.TipoTrabajo"><option
42              <span id="errorMgspan" asp-validation-for="TipoTRabajo" class="text-danger"></span>
43          </div>
```

Figura 4.62. Formulario para agregar una nueva tarea.

4.6.6 Front-End: Implementación de la función para la asignación de una tarea

En la figura 4.63 se muestra la implementación para enviar el **id** de la **tarea** que será asignada, en la línea 378 de código javascript al botón con el identificador ***IdAsignarT*** tiene el atributo **onClick** que responderá al método ***AsignarTarea*** (ver indicador 1) cuando se selecciona el botón. El método ***AsignarTarea*** (ver indicador 1) cuando se ejecuta, invoca un método **swal** (ver indicador 2)proporcionado por **JQuery** que permite mostrar una alerta preguntando si está seguro de realizar la acción, en caso de una confirmación positiva se ejecuta la función siguiente que contiene un método **AJAX** (ver indicador 3), el cual envía el **id** de la tarea almacenado en el parámetro **e** (ver indicador 1) a la url ***/Tareas/Asignar/*** (ver indicador 4), donde el controlador es ***Tareas*** y la acción es ***Asignar*** a través del método **GET** (ver indicador 5). Cuando la acción

Asignar (**Back-End**) realizar su función y se ejecuta correctamente devuelve un **JSON** que contiene una propiedad llamada ***status*** (línea 398) de tipo booleano, en caso de que el valor de la propiedad sea verdadero se mostrará otra alerta notificando al usuario que se realizó correctamente la acción (ver indicador 6) y se refresca el ***treeview*** (ver indicador 7)**.**

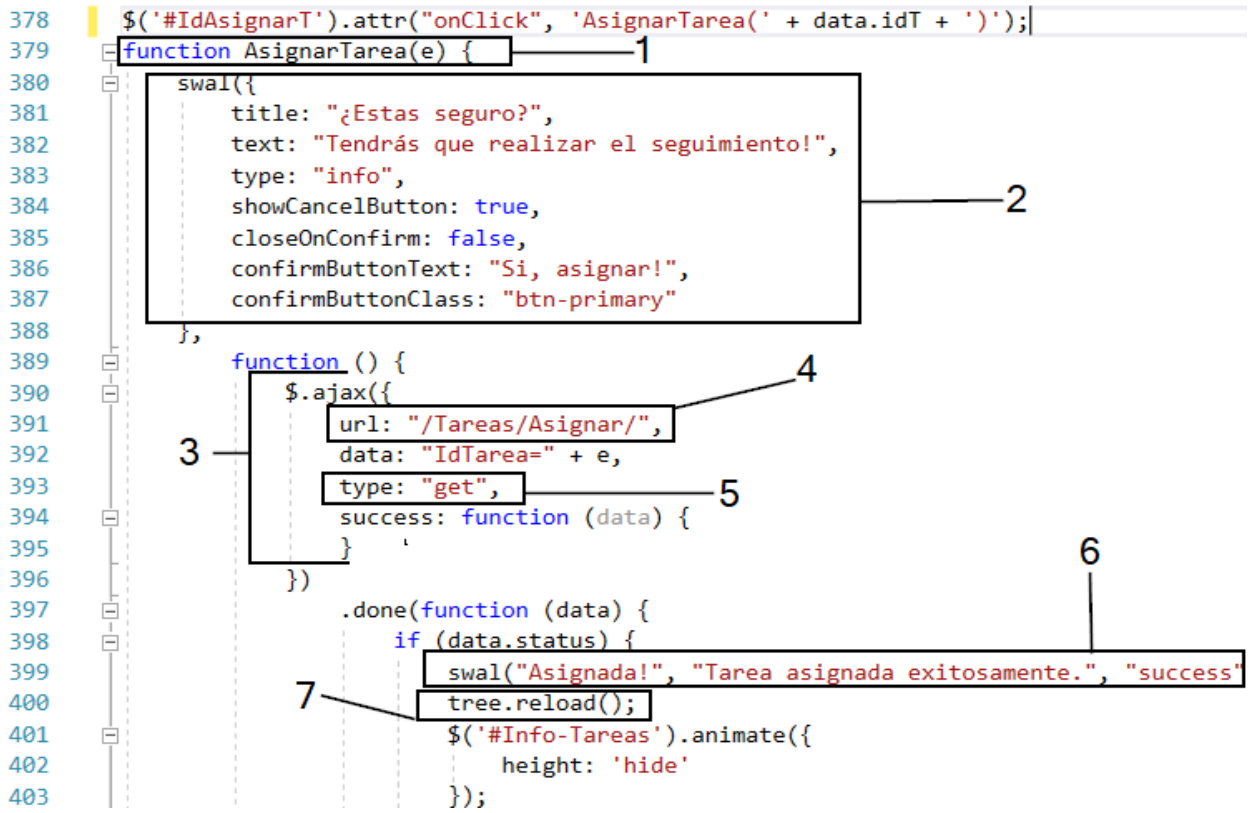


Figura 4.63. Función javascript para enviar el id de la tarea a la acción Asignar.

4.7 Sprint 5: Realización del seguimiento del Sprint

En este Sprint se presenta el desarrollo de las funciones para realizar que los usuarios con rol de equipo de desarrollo puedan registrar el tiempo invertido en las tareas asignadas (seguimiento del Sprint) y la implementación del burndown chart para la graficación del seguimiento de Sprints.

En la tabla 4.9 se presenta la planeación del Sprint 5, donde cada historia de usuario se desglosa en tareas que tienen un tiempo estimado para su realización, así como el inicio y finalización de este Sprint.

Tabla 4.9. Planeación del Sprint 5.

Fecha de inicio		**Fecha de finalización**		**N° Semanas**	
12-11-2019		26-11-2019		2	
Historia de usuario	**Tarea**	**Fecha de inicio**	**Fecha de finalización**	**Horas de trabajo de la tarea**	**Horas de trabajo total de la historia de usuario**
HU 17	Implementar una función que permita obtener las tareas asignadas a un integrante del equipo de desarrollo.	12-11-2019	14-11-2019	20	**64**
	Desarrollar un vista que permita visualizar a los usuarios con rol de equipo de desarrollo las tareas asignadas.	14-11-2019	15-11-2019	12	
	Crear un método que proporcione el mecanismo para registrar el tiempo invertido en las tareas asignadas a los integrantes del equipo de desarrollo.	16-11-2019	19-11-2019	26	
	Implementar una vista que proporcione un formulario para que los	19-11-2019	19-11-2019	6	

	usuarios con rol de equipo de desarrollo, puedan registrar el tiempo invertido en las tareas asignadas.				
HU 18	Desarrollar un método para obtener el tiempo invertido en las tareas de cada historia de usuario dentro de un Sprint.	20-11-2019	23-11-2019	32	**56**
	Implementar una vista para graficar el tiempo invertido en las tareas de cada historia de usuario dentro de un Sprint.	24-11-2019	26-11-2019	24	
Total de horas de trabajo invertidas					**120**

A continuación, en los siguientes apartados se presenta el desarrollo de las historias de usuario y tareas presentadas en la tabla 4.9, que abarcan la implementación para el registro del tiempo invertido en las tareas asignadas a los integrantes del equipo de desarrollo y su graficación.

4.7.1 Back-End: Implementación de la función para registrar el tiempo invertido a las tareas (seguimiento del Sprint)

Para el registro del tiempo que invierte un integrante del equipo de desarrollo se implementó la acción ***RegistrarTiempo*** con sus respectivos métodos **GET** y **POST** (ver figura 4.64).

```
217     public async Task<IActionResult> RegistrarTiempo(ulong? IdTarea)
218     {
219         TareasViewModel tvm = new TareasViewModel(unidadDeTrabajo,mapper);
220         if (IdTarea == null)
221             return StatusCode(400);
222         var ModeloT = await tvm.ObtenerTareaAsignada(IdTarea);
223         if (ModeloT == null)
224             return StatusCode(404);
225         return View(ModeloT);
226     }
227
228     [HttpPost]
229     [ValidateAntiForgeryToken]
        0 referencias | 0 solicitudes | 0 excepciones
230     public async Task<IActionResult> RegistrarTiempo(TareasViewModel tvm)
231     {
```

Figura 4.64. Acción GET y POST para registrar el tiempo invertido en una tarea.

La acción ***RegistrarTiempo*** en el método **GET** recibe como parámetro el id (***IdTarea***) de una tarea (línea 217), que se envía como argumento al método ***ObtenerTareasAsignadas*** (línea 222), el cual obtiene de la base de datos información de una tarea y esta es almacenada en la variable ***ModeloT*** de tipo ***TareasViewModel***. Después se obtener la información de la tarea, se envía a la vista con la estructura del ViewModel ***TareasViewModel*** (línea 225). La acción ***RegistrarTiempo*** se ejecuta cuando el usuario con rol de equipo de desarrollo selecciona una tarea asignada para registrar el tiempo invertido en ella, para que posteriormente la plataforma le proporcione un formulario para introducir el tiempo invertido y para ello es necesario enviar datos de la tarea selecciona a la vista sin necesidad de que el usuario visualice los datos de la tarea, entonces de esta manera el dato proporcionado por el usuario del tiempo invertido será almacenado en el ViewModel que fue enviado a la vista desde la acción ***RegistrarTiempo*** por el método **GET**.

Para el procedimiento del método **POST** se explicará en los siguientes tres pasos:

Paso 1: Obtención de los modelos

El método **POST** de la acción ***RegistrarTiempo*** (ver figura 4.65) recibe como parámetro un ViewModel llamado ***TareasViewModel*** almacenado en ***tvm***, el cual contiene los datos los datos para realizar el registro del tiempo invertido en una tarea que son proporcionados por el usuario con rol de equipo de desarrollo a través un formulario construido en el **Front-End**.

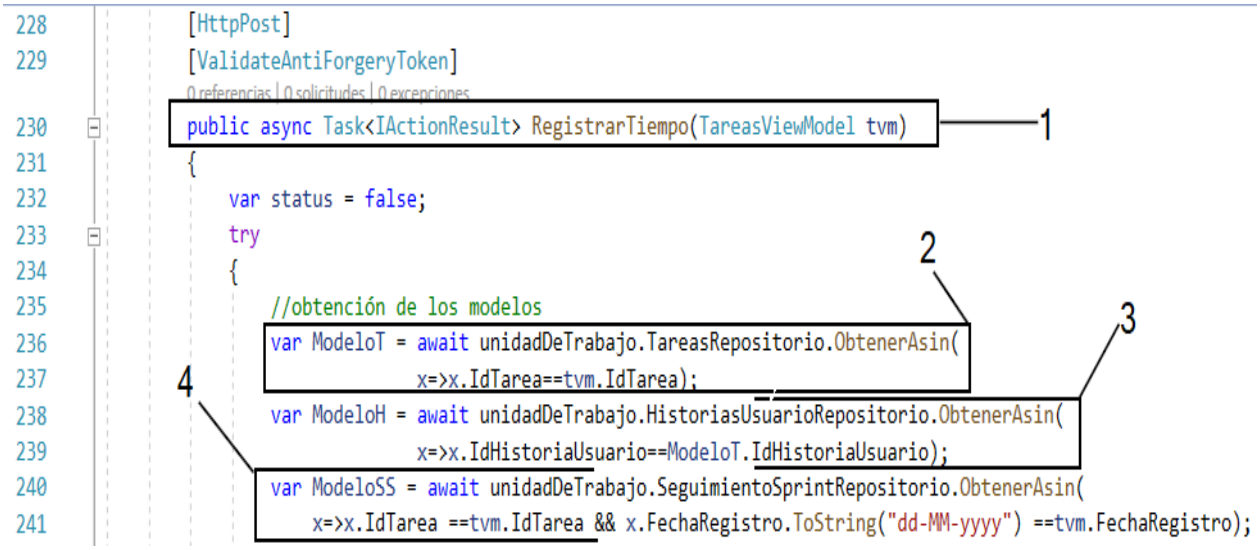


Figura 4.65. Obtención de los modelos Tareas, HistoriasUsuario y SeguimientoSprint.

Utilizando la información almacenada en el parámetro ***tvm*** se obtiene tres modelos ***Tareas*** (ver indicador 2), ***HistoriasUsuario*** (ver indicador 3) y ***SeguimientoSprint*** (ver indicador 4) realizando consultas **LINQ** a las tablas correspondientes a los modelos mencionados (ver líneas 236 a 241). En la consulta **LINQ** para la obtención del modelo ***SeguimientoSprint*** se realiza un filtrado para saber si existe un **registro de tiempo invertido** en la **fecha introducida** por el **usuario**, debido a que no puede existir más de un registro de tiempo invertido en una tarea en un día, y con esa consulta **LINQ** (ver indicador 4) se determina lo expuesto, si la consulta retorna nulo quiere decir que no existe un registro de tiempo invertido en la tarea en una fecha específica. Los modelos obtenidos con las consultas son almacenados en las variables ***ModeloT***, ***ModeloH***, ***ModeloSS***

(líneas 236, 238 y 240). Con la obtención de los tres modelos se podrá hacer el registro del tiempo invertido en una tarea.

Paso 2: Registro del tiempo invertido en una tarea

Con la obtención de los modelos se valida una condición (ver indicador 1 en la figura 4.66), donde si ***ModeloSS*** es nulo y el estatus de la tarea está **PENDIENTE** o en **PROCESO**) se realiza un mapeo entre las clases ***TareasViewModel*** y ***SeguimientoSprint*** (ver indicador 2) para obtener un modelo de tipo ***SeguimientoSprint*** almacenado en la variable ***Modelo***.

```
if (ModeloSS == null && (ModeloT.Estatus=="PENDIENTE" || ModeloT.Estatus=="PROCESO"))
{
    var Modelo=mapper.Map<TareasViewModel, SeguimientoSprint>(tvm);
    //validando si el tiempo invertido no sobre pasa el tiempo restante de la tarea
    if(ModeloT.EstimacionEsfuerzoRestante<Modelo.TiempoInvertido)
        return Json(data: new { status, accion = "Registrar_Tiempo",tiempoFaltante=ModeloT.Estim
    else
    {
        //realizando la resta al momento de registrar el tiempo
        ModeloT.EstimacionEsfuerzoRestante -= Modelo.TiempoInvertido;
        ModeloH.EstimacionEsfuerzoRestante -= Modelo.TiempoInvertido;
        //guardando el registro en la entidad del seguimiento del sprint
        Random r = new Random();
        Modelo.IdSeguimientoSprint= Convert.ToUInt64(r.Next(1, 100) + DateTime.Now.ToString("yyy
        await unidadDeTrabajo.SeguimientoSprintRepositorio.AgregarAsin(Modelo);
```

Figura 4.66. Registro del tiempo invertido en una tarea.

Posteriormente al mapeo, se valida si la **estimación de esfuerzo restante** de la **tarea** es menor al tiempo invertido (ver indicador 3) proporcionado por el usuario, en caso de ser válido esto, se envía un mensaje de advertencia al usuario que el tiempo de inversión introducido es mayor al faltante. Si esta condición es falta, entonces se realiza una resta en la **estimación de esfuerzo restante** de la **tarea** e **historia de usuario** (ver indicador 4) y por último se registra el tiempo invertido a través del método ***AgregarAsin*** (ver indicador 5).

Paso 3: Realizando cambios en las entidades Tareas e HistoriasUsuario

Como se puede observar en la figura 4.67, si la **estimación de esfuerzo restante** llega a 0 (ver indicador 1) en la **tarea** e **historia de usuario**, el estatus de cambia a **FINALIZADO** (ver indicador 2). Después de actualizar la propiedad Estatus de los modelos (***ModeloT*** y ***ModeloH***) se realizan los cambios en las tablas ***HistoriasUsuario*** y ***Tareas*** con los modelos almacenados en las variables ***ModeloH*** y ***ModeloH*** a través del método ***ActualizarAsin*** (ver indicador 3).

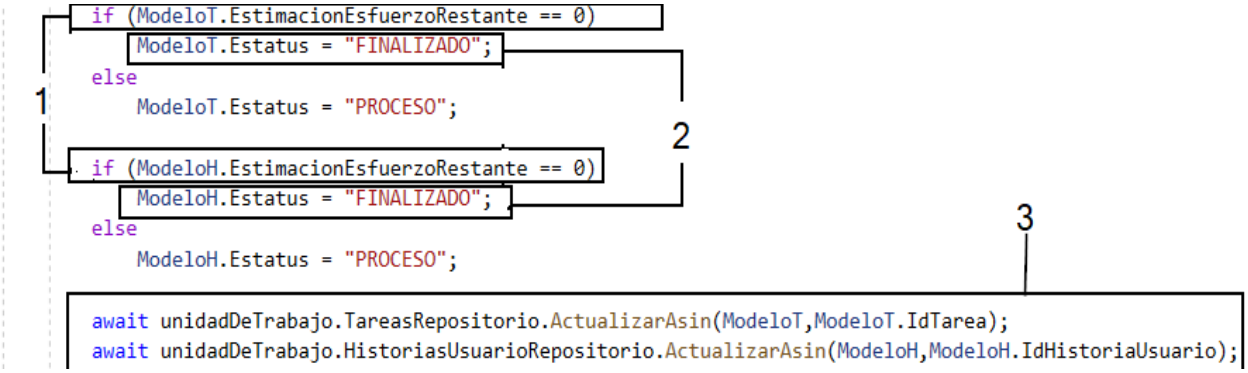


Figura 4.67. Actualización de las entidades Tareas e HistoriasUsuario.

4.7.2 Back-End: Implementación de las funciones para graficar el seguimiento del Sprint, historias de usuario y tareas

Para generar el **burndown chart** y este pueda proporcionar el progreso de un Sprint gráficamente se creó una acción llamada ***Graficar*** (ver figura 4.68). Donde esta acción proporciona datos a la vista para que grafique el seguimiento del Sprint a nivel **tareas**, **historias de usuario** y **Sprint**. Cuando se cumple una de las tres condiciones (líneas de código 43, 48 y 50) este invoca un método (***SeguimientoTarea***, ***SeguimientoHistoriaUsuario*** o ***SeguimientoSprint***) a través del objeto **objSVW** que retorna un valor de tipo **tupla**, la cual contiene los datos que serán graficados en la vista, el valor máximo del eje Y, el nombre de la tarea, historia de usuario o sprint. La lista de datos para visualizar la gráfica, son almacenados en el primer componente (item1) de la tupla y es enviada a la vista (línea de código 61), en el segundo componente se almacena el nombre la

tarea, historia de usuario o sprint y en el tercer componente se almacena el valor máximo del eje Y.

```
39          public async Task<IActionResult> Graficar(ulong id,string tipo)
40          {
41              SeguimientoViewModel objSVM = new SeguimientoViewModel(unidadDeTrabajo, mapper);
42              Tuple<List<Coordenadas>, string, int?,string> source = null;
43              if (tipo == "TA")
44              {
45                  source = await objSVM.SeguimientoTarea(id);
46                  ViewBag.Usu = source.Item4;
47              }
48              else if (tipo == "HU")
49              {
50                  source = await objSVM.SeguimientoHistoriaUsuario(id);
51                  ViewBag.Usu = "";
52              }
53              else if (tipo == "SP")
54              {
55                  source = await objSVM.SeguimientoSprint(id);
56                  ViewBag.Usu = "";
57              }
58              ViewBag.Nombre = source.Item2;
59              ViewBag.MaxEjeY = source.Item3;
60              ViewBag.tipo = tipo;
61              return View(source.Item1);
62          }
```

Figura 4.68. Acción para obtener los datos del seguimiento del Sprint, tarea o historia de usuario.

4.7.3 Front-End: Implementación de la vista para graficar el seguimiento a nivel Sprint, historia de usuario y tareas

Los datos enviados a la vista ***Graficar.cshtml*** (ver figura 4.69) desde la acción ***Graficar*** implementada de lado del servidor, a través de **razor** se crearon las variables **XLabels** y **YValues** (líneas 9 y 10) para almacenar las coordenadas que serán graficadas.

Graficar.cshtml*

```
1   @model List<Coordenadas>
2   @{Layout = null;}
3   @{ViewData["Title"] = "Seguimiento";}
4   @{
5       var XLabels = Newtonsoft.Json.JsonConvert.SerializeObject(Model.Select(x => x.x).ToList());
6       var YValues = Newtonsoft.Json.JsonConvert.SerializeObject(Model.Select(x => x.y).ToList());
7       var XLabelsP = "";
8       var YLabelsP = "";
9       var sp = ViewBag.lstP;
10      ViewData["Title"] = "Line Chart";
11      var Titulo = "";
12      if (ViewBag.tipo == "TA")
13      {
14          Titulo = Convert.ToString("Horas de trabajo invertidas por " + @ViewBag.Usu + " en la t
15      }
16      else if (ViewBag.tipo == "HU")
17      {
18          Titulo = Convert.ToString("Horas de trabajo invertidas en la Historia de Usuario " + @V
19      }
20      else if (ViewBag.tipo == "SP")
21      {
22          Titulo = Convert.ToString("Horas de trabajo invertidas en el sprint " + @ViewBag.Nombre
```

Figura 4.69. Creación de variables para la graficación del seguimiento.

Posterior a esto se creó una variable llamada **Titulo** (línea 11) que almacena el título de la gráfica donde esta cadena contiene los datos enviados desde el servidor a través de los **ViewBag** (líneas 12, 16 y 20) como se observa en la figura 4.69.

Después de haber creado las variables que contiene los datos necesarios para visualizar la gráfica se realizó la maquetación HTML donde a través de la etiqueta **canvas** (línea 30) se visualizará la gráfica, para ello se le asignó un id con clave **chart** (ver figura 4.70). Después de haber creado la maquetación se manda a llamar la librería **chart js** y **JQuery** (líneas 35 y 36) que proporcionar los mecanismos necesarios para visualizar la gráfica.

```
22  <html>
23  <head>
24      <meta name="viewport" content="width=device-width" />
25      <title>Line</title>
26  </head>
27  <body>
28      <div class="box-body">
29          <div class="chart-container">
30              <canvas id="chart" style="width:70%; height:500px"></canvas>
31          </div>
32      </div>
33  </body>
34  </html>
35  <script src="https://cdnjs.cloudflare.com/ajax/libs/Chart.js/2.7.2/Chart.bundle.min.js"></script>
36  <script src="https://code.jquery.com/jquery-3.3.1.min.js"></script>
```

Figura 4.70. Maquetación de la vista para graficar el seguimiento.

En el mismo documento correspondiente a la vista ***Graficar.cshtml*** se declaró un área javascript que contiene las opciones de configuración, estilo y funcionalidad para visualizar la gráfica con los mecanismos proporcionados por **chart js** (ver figura 4.71). Como se muestra en la figura 4.71 se obtiene el objeto de la etiqueta **canvas** a través del método **getElementById** donde se necesita como argumento el **id** del objeto del DOM (línea 41). Este objeto obtenido es almacenado en la variable ***ctx*** (línea 41), posteriormente se declara una variable llamada ***data*** (línea 42) en formato **JSON** que contiene la configuración para visualizar las etiquetas en el **eje x** (propiedad ***labels***) proporcionadas por la variable ***XLabel*** declarada al principio del documento de la vista **Graficar.*cshtml***. En la misma estructura **JSON** se declara una propiedad **datasets** donde se establece un texto para la etiqueta (línea 45) que aparecerá cuando se pase el cursor en los puntos graficados, además se establece el color de la línea que une a los puntos (línea 48) y lo más importante los datos en el eje Y para graficar los puntos con respecto a ese eje X, estos datos son proporcionados por la variable ***YLabels*** (línea 51).

```
38  <script type="text/javascript">
39
40          $(function () {
41          var ctx = document.getElementById("chart").getContext('2d');
42          var data = {
43                  labels: @Html.Raw(XLabels),
44                  datasets: [{
45                      label: "Tiempo restante",
46                      fill: false,
47                      lineTension: 0,
48                      backgroundColor:'#66b3ff',
49                      borderColor:'#66b3ff',
50                      borderWidth: 1,
51                      data: @Html.Raw(YValues)
52      }]
53              };
```

Figura 4.71. Configuración de los datos para visualizarlos en la gráfica del seguimiento.

Posterior al establecimiento de los datos, se configura la escala (líneas 61 a 83) y título de la gráfica (línea 59), además de las etiquetas para identificar con un nombre a cada eje en este caso el eje Y tendrá el nombre de ***Horas*** (línea 70) y eje X será ***Días*** (línea 83) (ver figura 4.72). Esta configuración de estilo se almacena en una variable llamada ***options*** (línea 55) en formato JSON.

```
55  var options = {
56                  maintainAspectRatio: false,
57                  title: {
58                      display: true,
59                      text:'@Titulo'
60                              },
61                  scales: {
62                      yAxes: [{
63                          ticks: {
64                              min: 0,
65                              max:@ViewBag.MaxEjeY,
66                              beginAtZero: true
67                          },
68                          scaleLabel: {
69                              display: true,
70                              labelString:'Horas'
71                          },
72                          gridLines: {
73                              display: true,
74                              color: "rgba(255,99,164,0.2)"
75                          }}],
76                      xAxes: [{
77                          ticks: {
78                              min: 0,
79                              beginAtZero: true
80                          },
81                          scaleLabel: {
82                              display: true,
83                              labelString:'Días'
```

Figura 4.72. Establecimiento del estilo de la gráfica.

Con el establecimiento de los datos, estilo y la obtención del objeto donde se visualizará la gráfica, estas configuraciones fueron almacenadas en las variables ***options***, ***data*** y ***ctx*** las cuales son pasadas como argumentos en el método **Chart** proporcionado por la librería **Chart.js** como se observa en la figura 4.73. En el argumento **type** del método **Chart** se indica el tipo de gráfica, en este caso es de tipo línea, debido a que el burndown chart se visualiza bajo este tipo de gráfica.

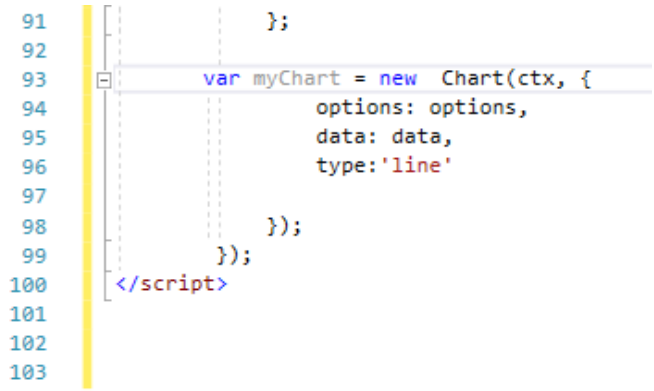


Figura 4.73. Paso de argumentos al método Chart para visualizar la gráfica.

Después de haber descrito el desarrollo de las funcionalidades de la plataforma propuesta en cada uno de los **Sprints** presentados en los apartados anteriores, en el siguiente apartado se presentan pantallas de la plataforma desarrolla que tiene como nombre ***CyScrum***, contrarrestando con algunos casos de uso presentados en el capítulo 3.

4.8 CyScrum vs Casos de uso

Para facilitar la comparativa entre los casos de uso y la plataforma ***CyScrum*** se presentan los mismos casos uso que fueron mostrados en el capítulo 3. En la figura 4.74 se presenta el caso de uso para realizar el registro de las historias de usuario, indicando que los usuarios con roles de **product owner**, **Scrum master** y **equipo de desarrollo** puede consultar la información de las historias de usuario, pero únicamente el usuario con rol de **product owner** puede agregar o registrar las **historias de usuario** en el **product backlog** de un proyecto.

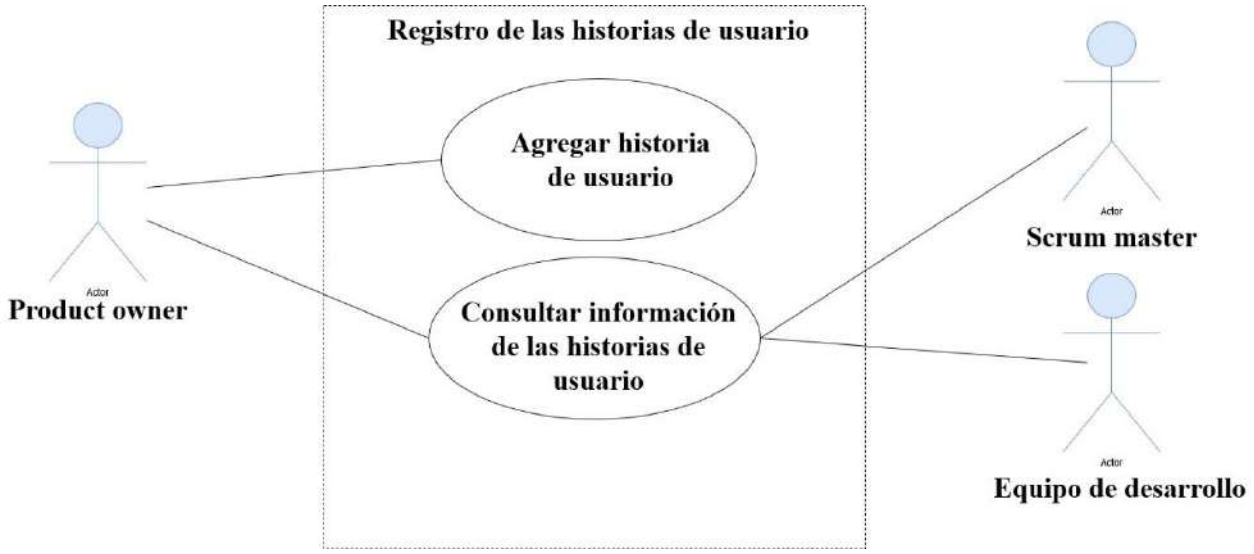


Figura 4.74. Caso de uso para el registro de las historias de usuario.

En la figura 4.75 se observa la página principal de la plataforma ***CyScrum***, donde se muestra el **product backlog** (ver indicador 2) de un proyecto a gestionar, en este caso el proyecto que se está gestionando con ***CyScrum*** tiene como nombre ***SISMEC*** (ver indicador 2). El usuario con nombre ***José Raúl López Morales*** en el proyecto ***SISMEC*** tiene el rol de **product owner** (ver indicador 1), por tener ese rol la plataforma ***CyScrum*** le habilita la opción de **Agregar Historia Usuario** (ver indicador 3), la cual permite al usuario con rol de **product owner**, registrar historias de usuario en el **product backlog**.

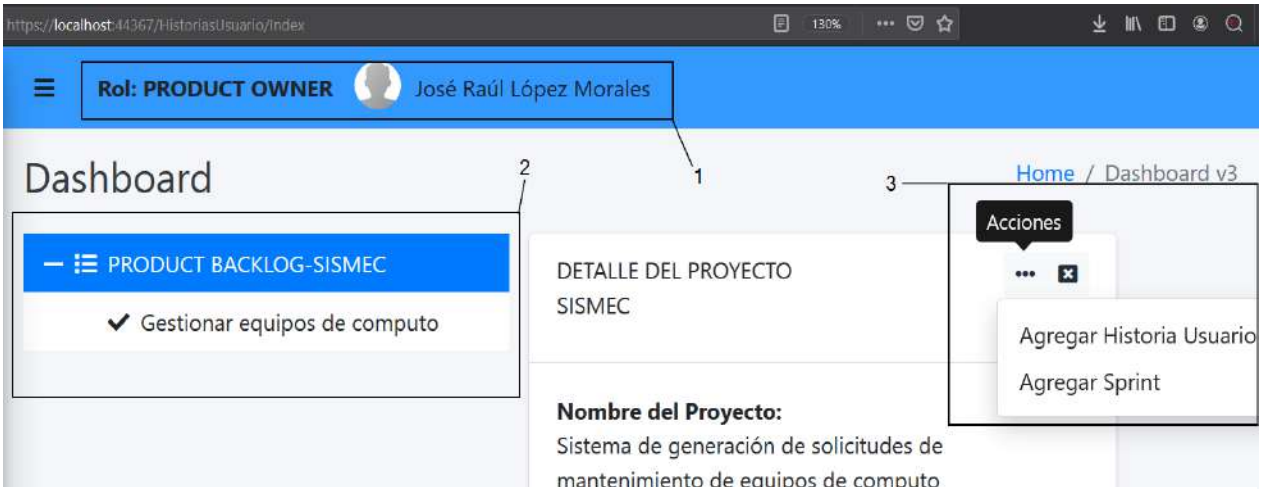


Figura 4.75. Vista principal de la plataforma CyScrum con un usuario con rol de product owner.

Cuando un usuario accede a un proyecto con rol de **equipo de desarrollo** (ver indicador 1 de la figura 4.76) la plataforma ***CyScrum*** no le habilita la opción **Agregar Historia Usuario** (ver indicador 3), solamente puede consultar las historias de usuarios del **product backlog** (ver indicador 2).

Figura 4.76. Vista principal de la plataforma CyScrum con un usuario con rol de equipo de desarrollo.

Con lo expuesto se puede concluir que la plataforma ***CyScrum*** cumple lo que establece visualmente el caso de uso **Registro de las historias de usuario** (ver figura 4.74). A continuación, se presenta otra comparativa entre el caso de uso **Gestión de las tareas** y la plataforma ***CyScrum***. En la figura 4.77 se observa el caso de uso **Gestión de las tareas**, indicando gráficamente que los usuarios con rol de **equipo de desarrollo** podrán agregar o registrar tareas a una historia de usuario,

así como consultar la información de las tareas y la asignación de estas. Los usuarios con rol de **Scrum master** y **product owner** solamente pueden consultar la información de las tareas.

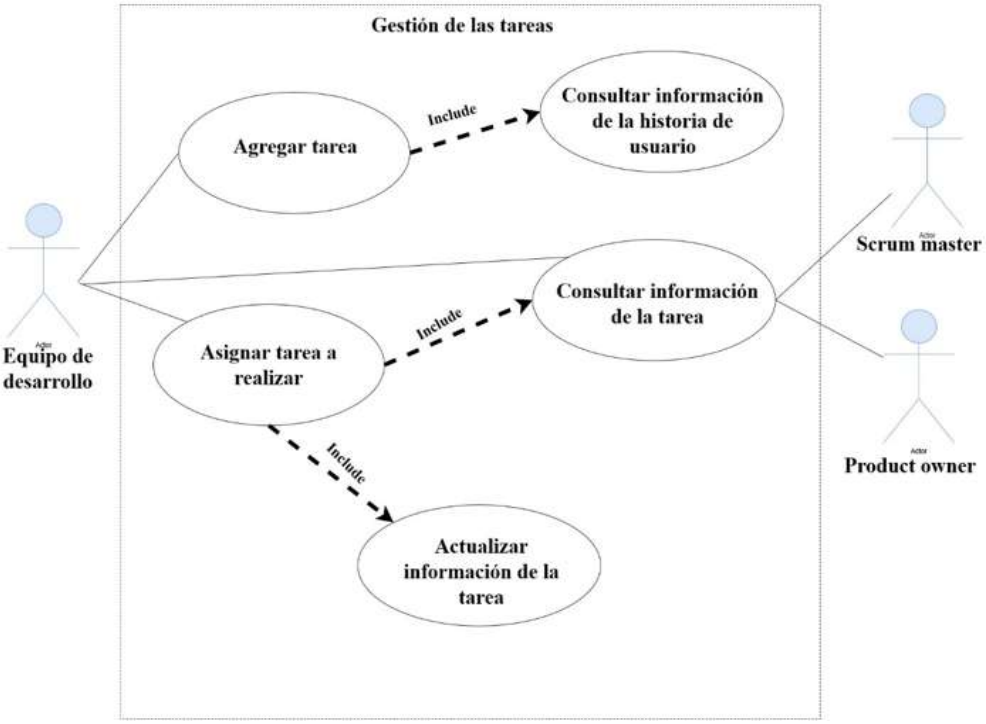


Figura 4.77. Casos de uso para la gestión de las tareas.

En la figura 4.78 se puede observar la plataforma ***CyScum***, donde accedieron dos usuarios que tiene diferentes roles, el usuario **José Alberto Medina Castañeda** con rol de **equipo de desarrollo** (ver indicador 1) tiene permitido agregar tareas a la historia de usuario **Gestionar usuarios** (ver indicador 2) del proyecto **SISMEC**. A diferencia del usuario **José Raúl López Morales** con rol de **product owner** (ver indicador 3) tiene permitido eliminar o modificar la historia de usuario **Gestionar usuarios** (ver indicar 4). Otro aspecto que se cumple con base al caso de uso **Consultar información de la historia de usuario** (ver figura 4.77), es que para agregar una tarea necesariamente se consulta la información de la historia de usuario.

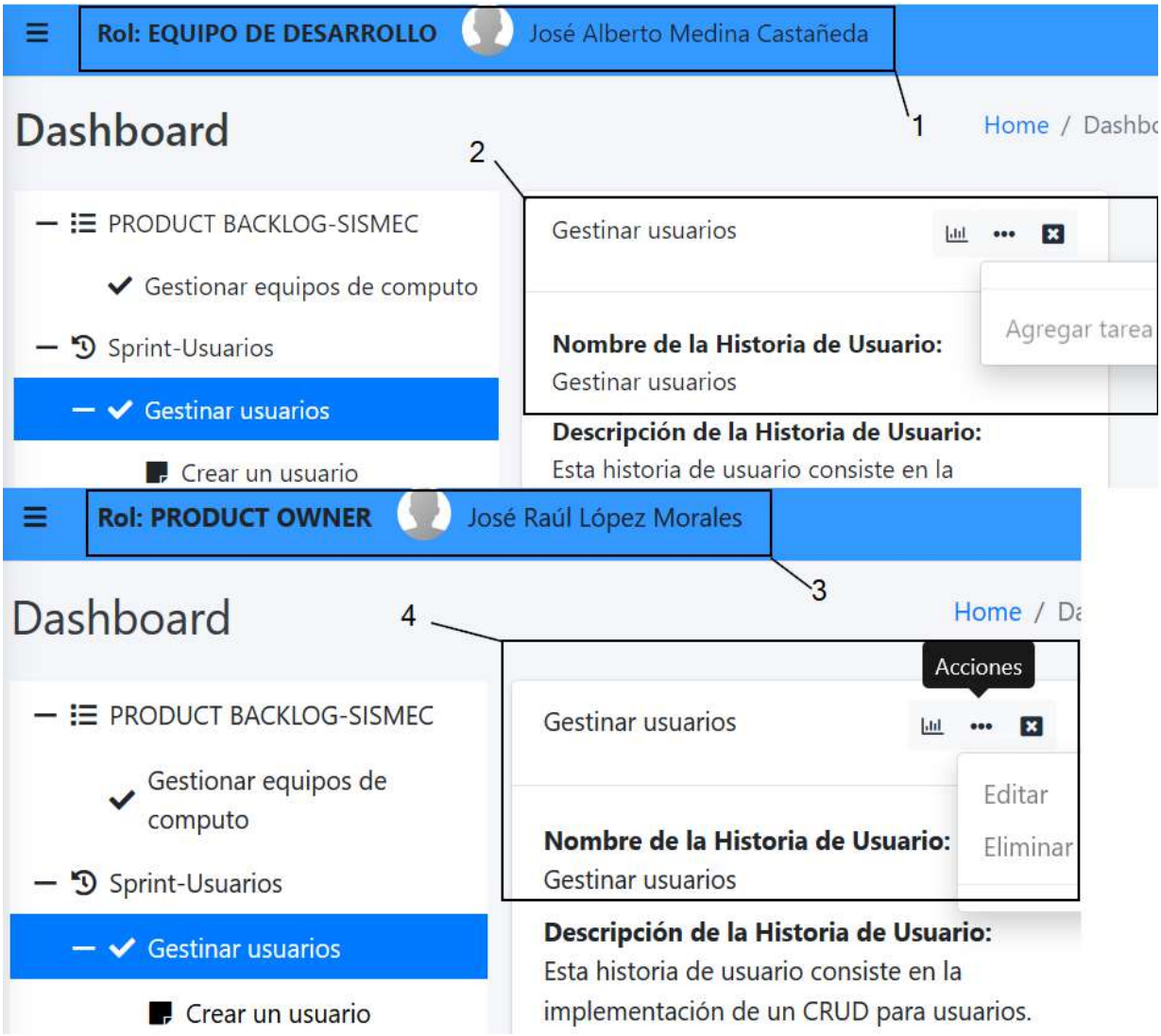


Figura 4.78. Comparativa entre las funcionalidades permitidas a los usuarios con base a su rol en la plataforma.

En la figura 4.79 se muestra la plataforma ***CyScrum***, donde accedieron dos usuarios que tiene diferentes roles, el usuario **José Alberto Medina Castañeda** con rol de **equipo de desarrollo** (ver indicador 1) tiene permitido asignarse la tarea **Crear un usuario** (ver indicador 2) incluso puede eliminarla o modificar su información. Cuando una tarea no está asignada a un integrante del equipo de desarrollo aparecerá en la información de la tarea como **NO ASIGNADA** (ver indicador 4). A diferencia del usuario **José Raúl López Morales** con rol de **product owner** únicamente puede consultar la información de la tarea **Crear un usuario** (ver indicador6) y ver la lista de tareas del product backlog (ver indicador 7).

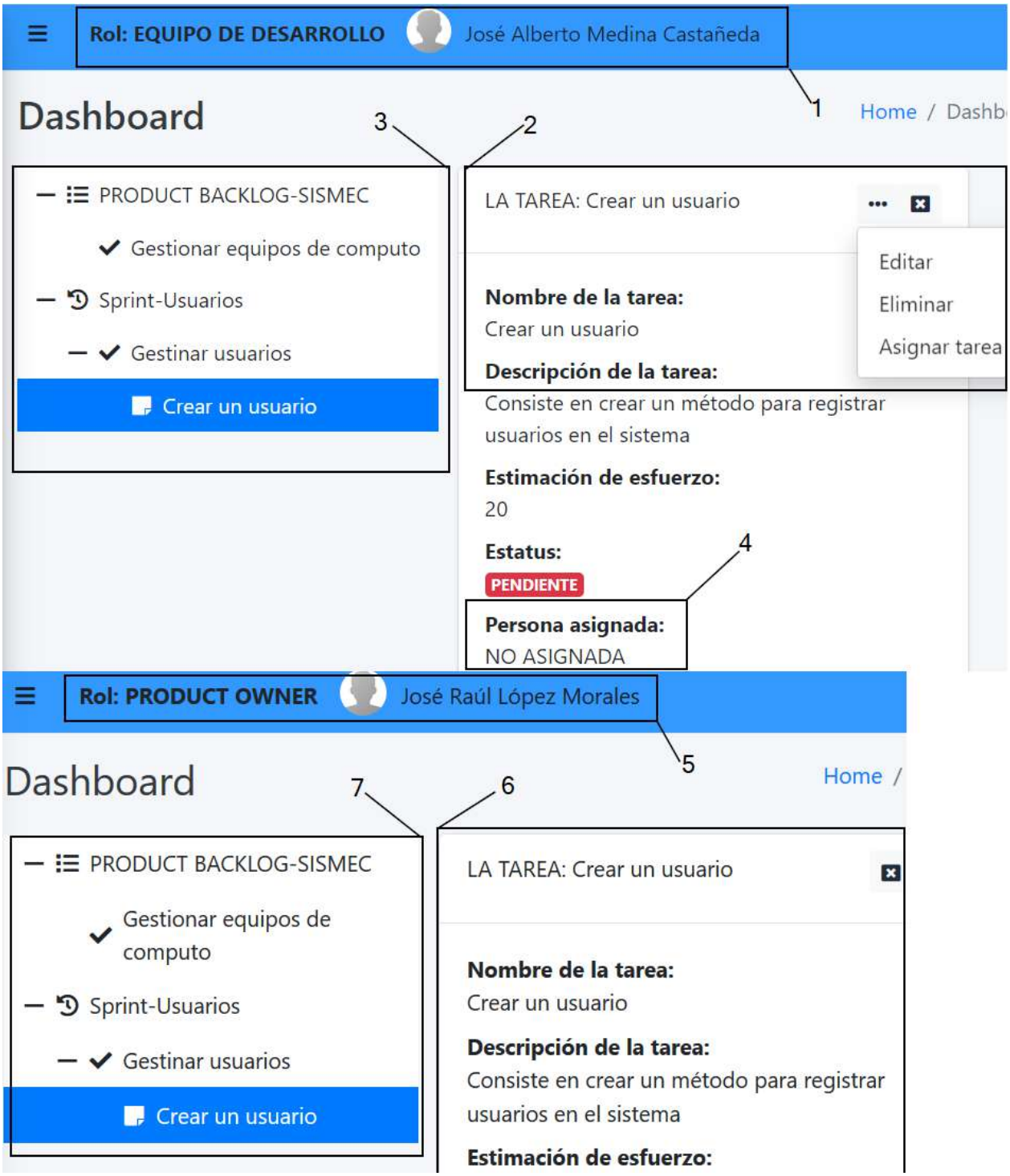


Figura 4.79. Comparativa entre de las vistas con dos usuarios de diferentes roles.

Para comprobar el caso de uso **Actualizar información de la tarea** (ver figura 4.77), en la figura 4.80 se muestra la plataforma ***CyScrum***, donde el usuario **José Alberto Medina Castañeda** con rol de **equipo de desarrollo** (ver indicador 1) se asignó la tarea **Crear un usuario** (ver

indicador 2) y entonces se le habilita la opción **Desasignar tarea**. Cuando un usuario se asigna una tarea la información de una tarea se actualiza, en este caso la tarea **Crear un usuario** está asignada al usuario **José Alberto Medina Castañeda** (ver indicador 3). En conclusión, cuando se asigna una tarea necesariamente se tiene que actualizar la información de esta y esto se puede observar en la figura 4.80.

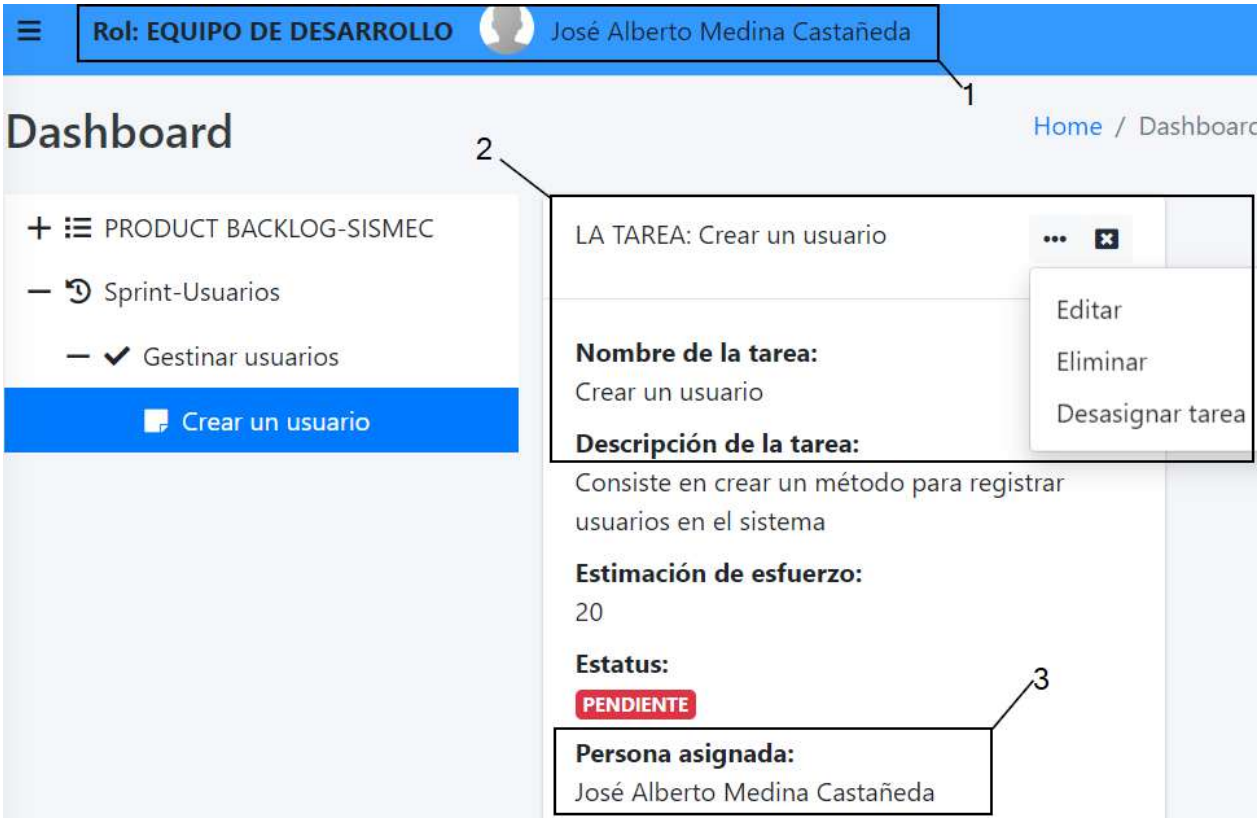


Figura 4.80. Asignación de una tarea en la plataforma CyScrum.

Una vez codificadas las funciones de la plataforma ***CyScrum*** es posible probar su funcionamiento. En el siguiente capítulo se muestran los resultados del funcionamiento del trabajo realizado aplicando a los diferentes casos de estudio la gestión y monitorización de proyectos de desarrollo de software basados en la metodología Scrum.

Capítulo 5 Resultados

En este capítulo se presentan los resultados del trabajo y como se han obtenido a partir de cuatro casos de estudio correspondientes a las funciones desarrolladas de la plataforma ***CyScrum*** en los Sprints presentados en el capítulo 4. El análisis de los resultados obtenidos permitirá definir las conclusiones que se obtengan y una mejor comprensión para los trabajos de investigación derivados del tema.

Antes de presentar los casos de estudio es necesario describir la inicialización del proyecto para que posteriormente se explique cada una de las funciones proporcionadas por la plataforma ***CyScrum***.

5.1 Inicializando la plataforma CyScrum

Para inicializar instancia ***PiScrum_2_0*** (ver indicador 2 de la figura 5.1) de la plataforma ***CyScrum*** se hace puede hacer a través del IDE Visual Studio, simplemente seleccionando el botón **IIS Express** (ver indicador 1). Visual Studio proporciona un servidor web IIS en una versión más ligera para pruebas y desarrollo de aplicaciones web.

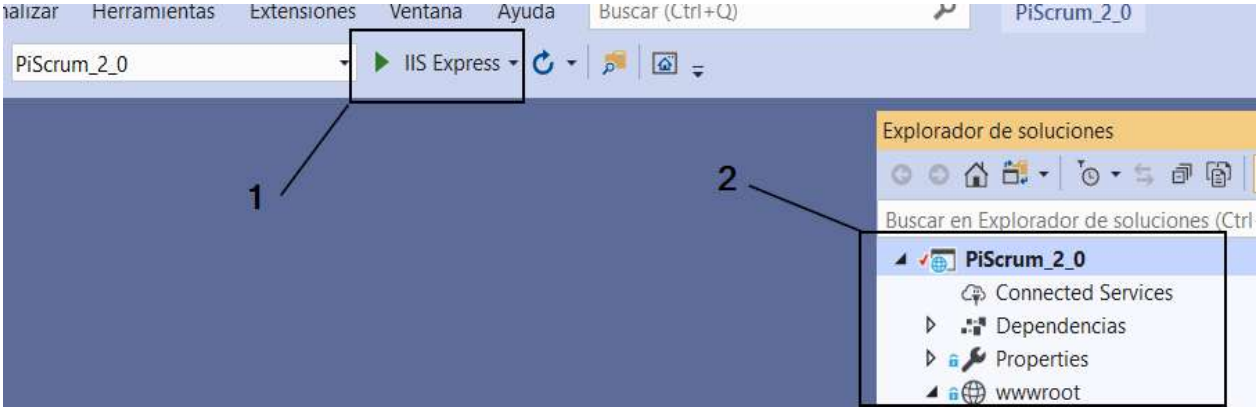


Figura 5.1. Instancia de la plataforma CyScrum en Visual Studio.

Cuando se selecciona el botón **IIS Express**, Visual Studio comenzará a ejecutar **IIS Express**, donde a su vez **IIS Express** carga las **dll** de la instancia ***PiScrum_2_0***, como se observa en el panel de salida de Visual Studio (ver figura 5.2).

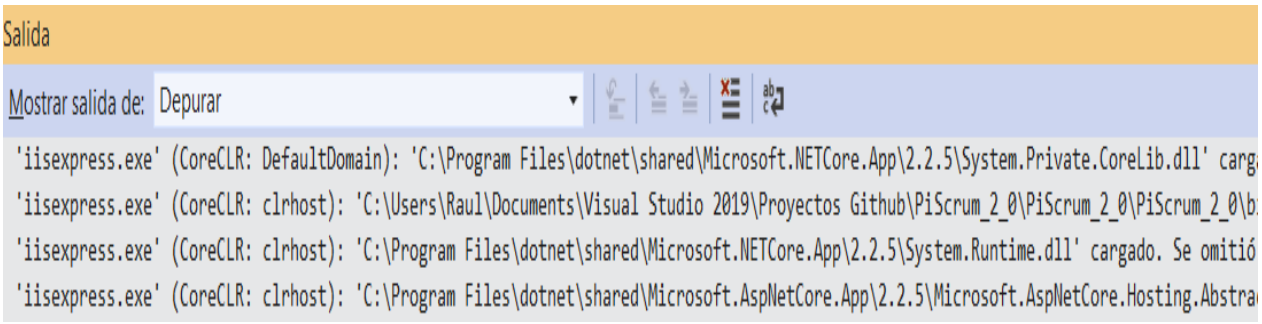


Figura 5.2. Salida de depuración y compilación de la instancia PiScrum_2_0.

Cuando vez terminada la compilación, se ejecuta la aplicación y automáticamente se mostrará la vista de inicio de sesión de la plataforma ***CyScrum*** en un navegador web que este configurado como predeterminado en el sistema operativo, en este caso es FireFox Mozilla (ver figura 5.3).

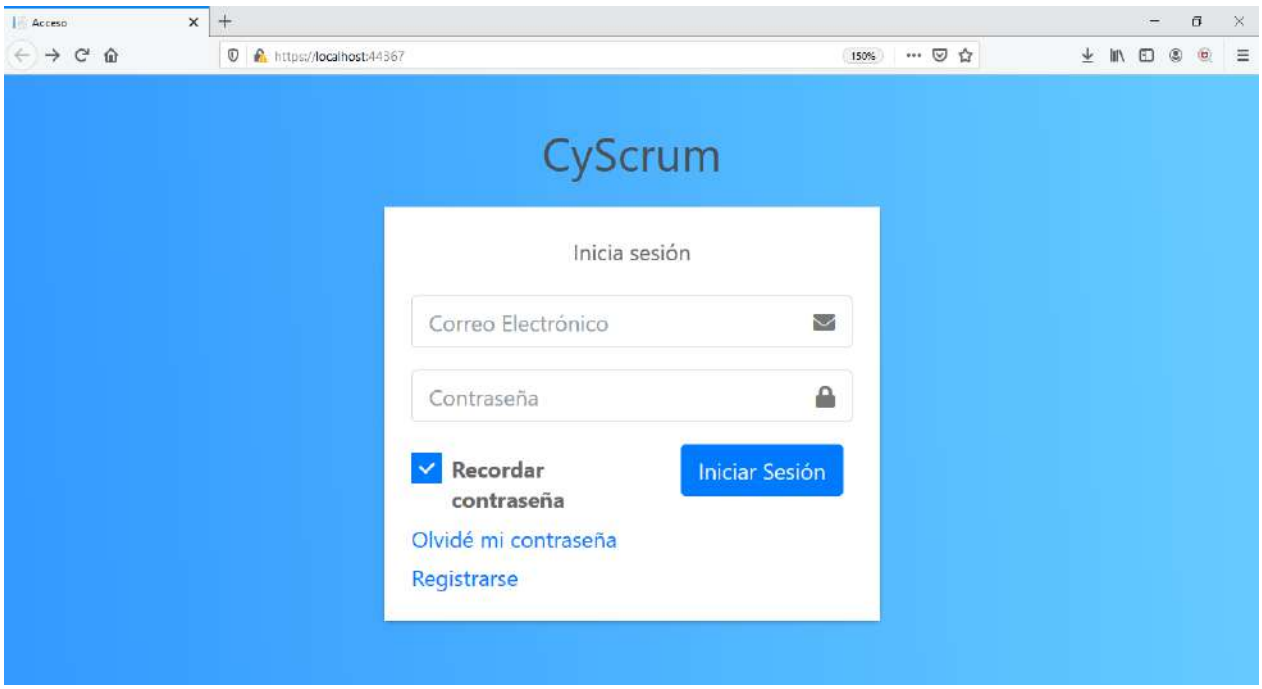


Figura 5.3. Inicio de sesión de la plataforma CyScrum.

A partir de aquí es posible iniciar la evaluación de los resultados de desempeño de la plataforma ***CyScrum***. La estructura de cada uno de los casos de estudio, está compuesta por una descripción

general del caso de estudio, los objetivos que se pretenden alcanzar, el procedimiento que consiste en una serie de pasos a realizar en el caso de estudio para cumplir con los objetivos, posteriormente el desarrollo donde se explica cada paso del procedimiento y por último la conclusión de los casos de estudio.

5.2 Caso de estudio 1: Creación de usuarios

Descripción

En el presente caso de estudio, se realizará la creación de usuarios que accederán a la plataforma ***CyScrum*** con el fin de gestionar desarrollo basado en la metodología Scrum.

Objetivos

1. Realizar el registro de 2 usuarios para que puedan tener acceso a la plataforma ***CyScrum***.
2. Comprobar en la base de datos los registros de los usuarios creados.
3. Validar los correos electrónicos proporcionados por los usuarios creados para tener acceso a la plataforma ***CyScrum***.
4. Comprobar en la base de datos que los correos electrónicos de los usuarios creados estén validados.
5. Realizar el inicio de sesión con los dos usuarios creados.

Procedimiento

1. Abrir el navegador web FireFox Mozilla.
2. Dirigirse a https://localhost:44367/.
3. Crear un usuario en la plataforma ***CyScrum*** (**HU 1**).
4. Verificar la cuenta de correo electrónico del usuario creado (**HU 2**).
5. Iniciar sesión con el usuario creado.

6. Crear un segundo usuario e iniciar sesión en la plataforma ***CyScrum***.
7. Verificar en la base de datos los registros de los usuarios creados.

Desarrollo

1. Abrir el navegador web FireFox Mozilla.

En la computadora con sistema operativo Windows 10 se procede abrir el navegador web FireFox Mozilla, haciendo doble clic en el icono de aplicación (ver figura 5.4).

Figura 5.4. Icono del navegador web FireFox Mozilla.

2. Dirigirse a https://localhost:44367/.

En la barra de navegación del navegador web FireFox Mozilla se escribe la dirección https://localhost:44367 y se presiona la tecla [ENTER] (ver figura 5.5).

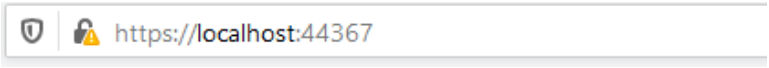


Figura 5.5. Barra de navegación de FireFox.

3. Crear un usuario en la plataforma ***CyScrum*** (**HU 1**).

Cuando se presiona la tecla [ENTER] la plataforma ***CyScrum*** presenta la vista para iniciar sesión, donde pide una dirección de correo electrónico (ver indicador 1 en la figura 5.6) y una contraseña (ver indicador 2). Para registrarse en la plataforma es necesario hacer clic en el hipervínculo **Registrarse** (ver indicador 3).

Figura 5.6.Vista de inicio de sesión para la plataforma CyScrum.

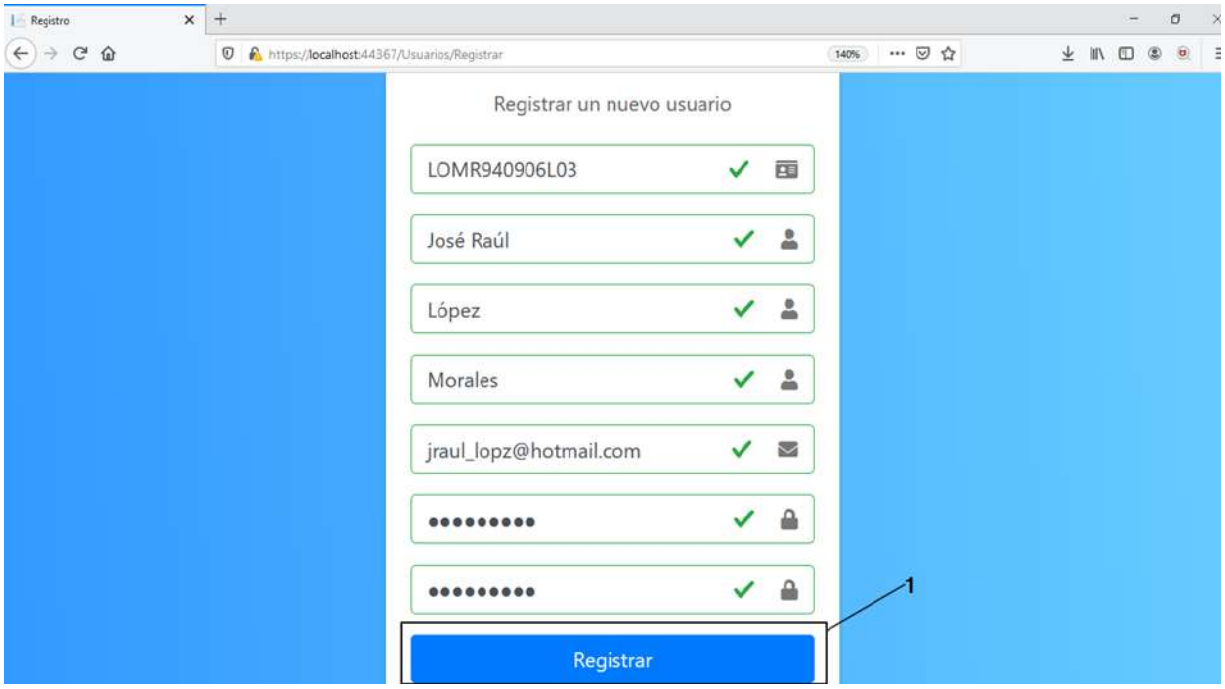


Figura 5.7. Vista para registrar un nuevo usuario en la plataforma CyScrum.

Cuando se hace clic en el hipervínculo **Registrarse**, la plataforma se re direcciona a la vista para registrar un nuevo usuario (ver figura 5.7). Esta vista proporciona un formulario donde solicita datos personales del usuario, una dirección de correo electrónico y contraseña para el acceso a la plataforma ***CyScrum***. Posteriormente al llenado del formulario, se

selecciona el botón **Registrar** (ver indicador 1) para realizar el registro de información en la plataforma ***CyScrum***.

Cuando se selecciona el botón **Registrar**, la plataforma ***CyScrum***, envía los datos al servidor y realiza el registro de los datos proporcionados. Posteriormente la plataforma ***CyScrum*** se re direcciona a una vista proporcionando un aviso para que el usuario registrado pueda validad la dirección de correo electrónico que introducido (ver figura 5.8). Este proceso se tiene que realizar antes de iniciar sesión, sino la plataforma denegará el acceso.

Figura 5.8. Vista de aviso para confirmar la cuenta de correo electrónico.

4. Verificar la cuenta de correo electrónico del usuario creado (**HU 2**).

La plataforma envía un correo electrónico a la dirección de correo electrónico proporcionada en el formulario **Registrar un nuevo usuario** (ver figura 5.7). El contenido del correo electrónico enviado por la plataforma ***CyScrum***, es dando indicaciones al usuario que para terminar el proceso de su registro, es necesario hacer clic en el hipervínculo

CONFIRMAR CORREO ELECTRÓNICO (ver indicador 1 en la figura 5.9) o copiar la dirección en un navegador web (ver indicador 2).

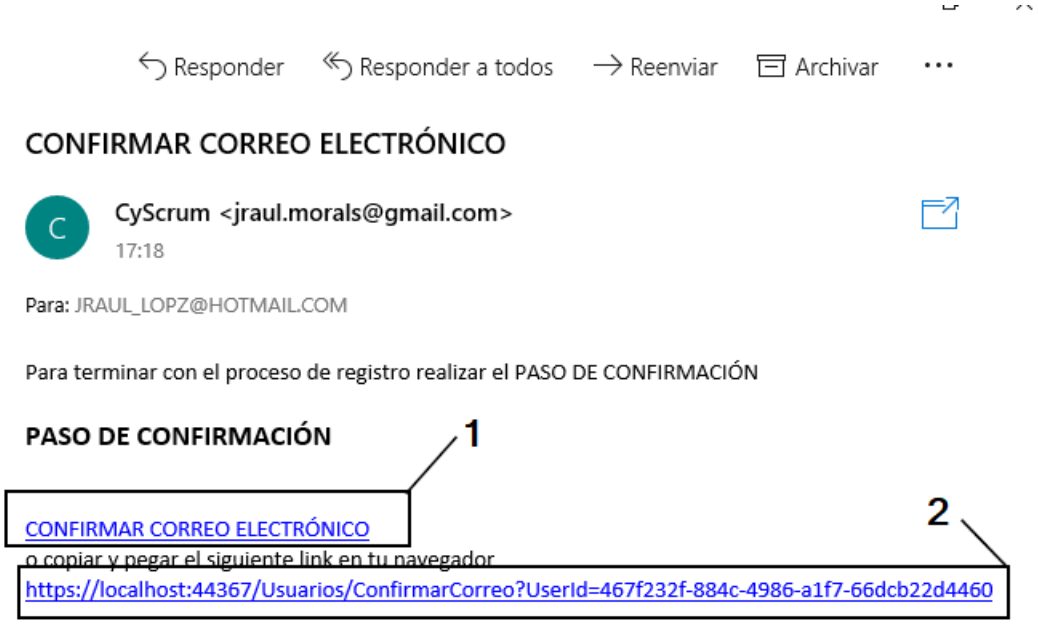


Figura 5.9. Correo electrónico para validar la cuenta de correo electrónico.

Figura 5.10. Vista con formulario para confirmar cuenta de correo electrónico.

Al hacer clic en el hipervínculo enviado por la plataforma ***CyScrum*** a la cuenta de correo electrónico proporcionada, automáticamente el sistema operativo abrirá una nueva ventana en el navegador, donde la plataforma ***CyScrum*** muestra una vista que contiene un aviso indicando al usuario que tiene que confirmar la dirección de correo electrónico para acceder

a su cuenta creada en la plataforma ***CyScrum*** (ver figura 5.10). Para confirmar la cuenta creada, simplemente se tiene que seleccionar el botón **Confirmar mi cuenta** (ver indicador 1).

5. Iniciar sesión con el usuario creado.

Después de haber seleccionado el botón **Confirmar mi cuenta**, la plataforma ***CyScrum*** se re direccionará a la vista de iniciar sesión (ver figura 5.11), donde tiene que introducir la dirección de correo electrónico y contraseña que proporciono el usuario con nombre **José Raúl** en el formulario **Registrar un nuevo usuario** (ver figura 5.7), para que posteriormente seleccione el botón **Iniciar sesión** (ver indicador 1 en la figura 5.11).

Figura 5.11. El usuario José Raúl introduciendo sus credenciales para iniciar sesión.

Cuando se haya iniciado sesión con el usuario **José Raúl**, la plataforma ***CyScrum*** lo re direccionará a la vista para seleccionar un proyecto, donde el usuario podrá ver su nombre completo (ver indicador 1 en la figura 5.12).

Figura 5.12. Vista para seleccionar un proyecto con sesión iniciada.

6. Crear un segundo usuario e iniciar sesión en la plataforma ***CyScrum***.

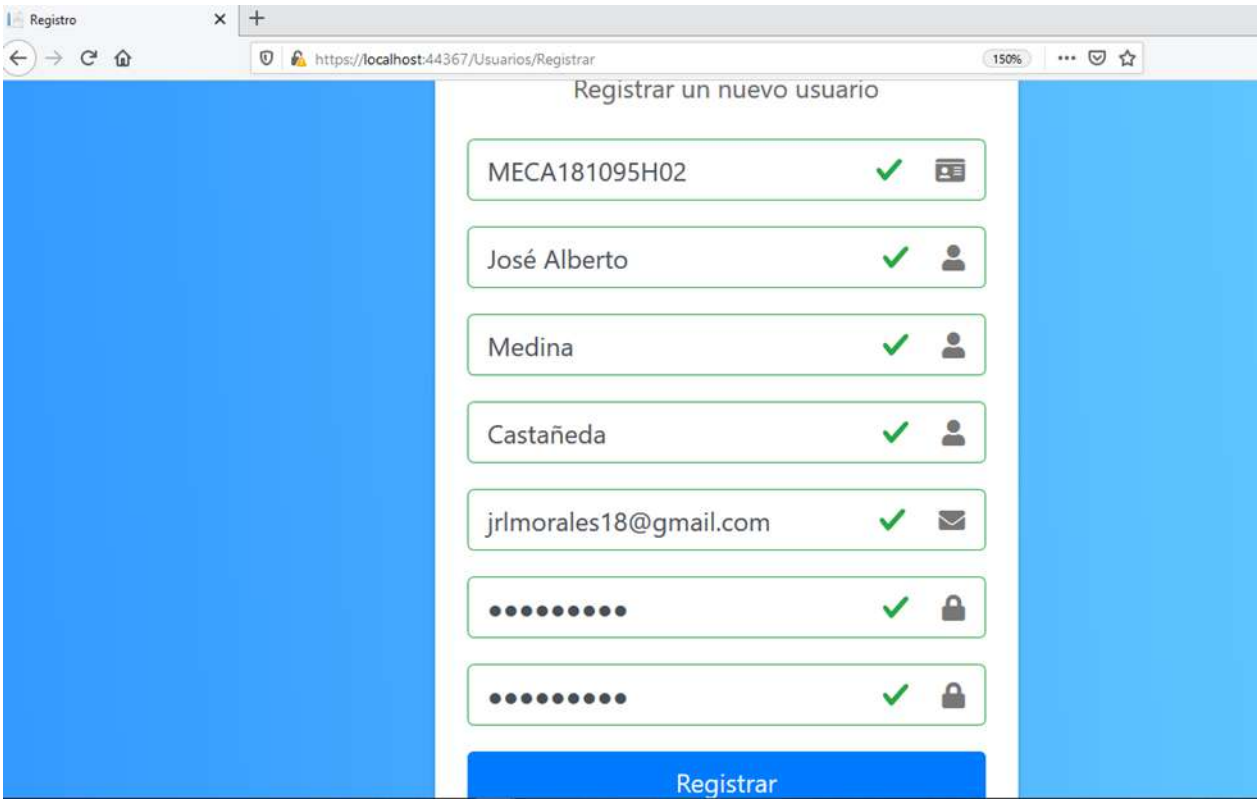


Figura 5.13. Vista para el registro de usuarios (usuario José Alberto).

Para la creación del segundo usuario se realiza el mismo proceso que se usó para crear al usuario **José Raúl**, entonces por tal motivo sólo se presentará la información que se proporcionó en el formulario para el registro de un nuevo usuario (**José Alberto**) en la plataforma ***CyScrum*** (ver figura 5.13), y su inicio de sesión como el usuario **José Raúl** (ver figura 5.14). Posteriormente a esto se verificarán los registros de los usuarios **José Raúl** y **José Alberto** en la base de datos.

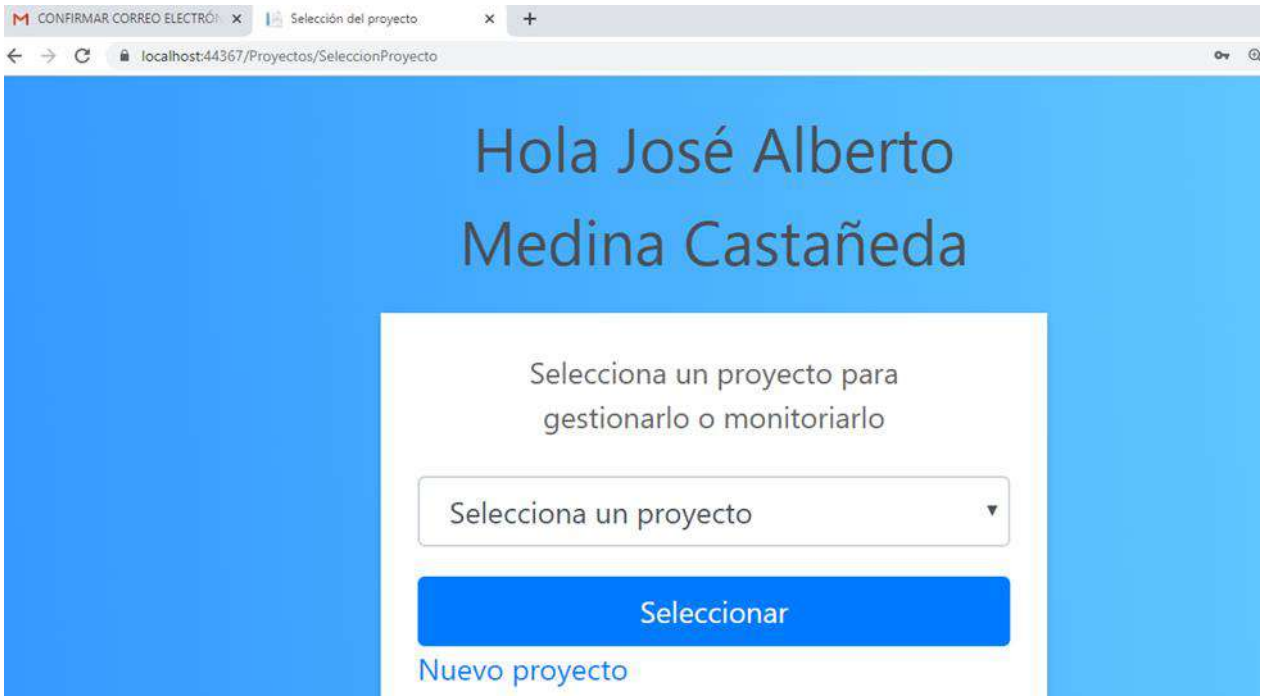
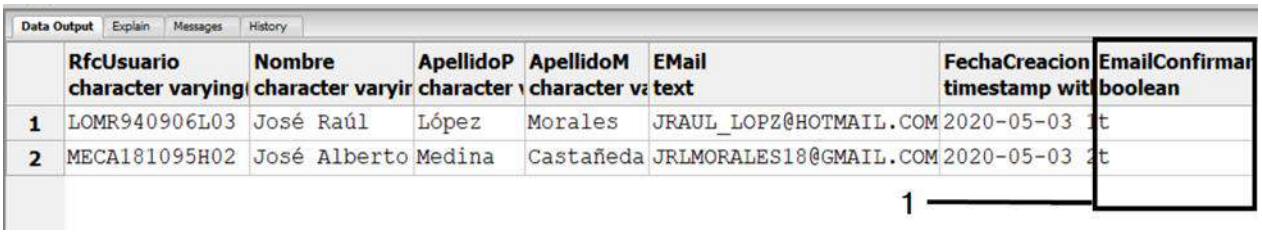


Figura 5.14. Inicio de sesión del usuario José Alberto en la plataforma CyScrum.

7. Verificar en la base de datos los registros de los usuarios creados.

Para verificar que la información de los usuarios (**José Raúl** y **José Alberto**) previamente creados en la plataforma ***CyScrum*** se hayan guardado correctamente, para ello se realizó una consulta a la tabla ***Usuarios*** de la base de datos de la plataforma ***CyScrum*** (ver figura 5.15).



Data Output | Explain | Messages | History

	RfcUsuario character varying(	Nombre character varyir	ApellidoP character v	ApellidoM character va	EMail text	FechaCreacion timestamp wit	EmailConfirmar boolean
1	LOMR940906L03	José Raúl	López	Morales	JRAUL_LOPZ@HOTMAIL.COM	2020-05-03 1	t
2	MECA181095H02	José Alberto	Medina	Castañeda	JRLMORALES18@GMAIL.COM	2020-05-03 2	t

1

Figura 5.15. Consulta de los usuarios registrados en la plataforma CyScrum.

En la figura 5.15 se puede observar que los usuarios creados en la plataforma ***CyScrum*** se guardaron correctamente, también se puede verificar que los usuarios confirmaron su dirección de correo electrónico (ver indicador 1). Es decir, cuando ha sido confirmada la dirección el campo ***EmailConfirmar*** (ver indicador 1) tiene el valor de **true** indicado con una **t** y en caso contrario tendrá el valor de false indicado con una **f**.

Conclusión

Se observa como resultado el registro de los dos usuarios en plataforma ***CyScrum***, siendo esto parte de los objetivos de este caso de estudio, además del cumplimiento con la **HU 1**, donde necesariamente los usuarios tienen que registrar en la plataforma ***CyScrum*** para gestionar desarrollo basado en la metodología Scrum. Con esto también se cumple con la **HU 2**, debido a que la plataforma envía un correo electrónico a los usuarios registrados para validar sus direcciones de correos electrónicos. De esta forma se cumplen con las **HU 1** y **HU 2** que se desarrollaron en el **Sprint 2** del capítulo 4.

5.3 Caso de estudio 2: Creación de proyectos

Descripción

En este caso de uso se realizará la creación de un proyecto para gestionarlo y monitorizarlo en la plataforma ***CyScrum*** con los usuarios previamente creados en el caso de estudio 1. Además, se presentará la forma de como agregar colaboradores con un rol a un proyecto. Por último, se mostrará una comparativa entre dos usuarios uno con rol de product owner y otro de equipo de desarrollo, donde se pueda observar los límites que tienen cada uno en un proyecto dependiendo de su rol.

Objetivos

1. Crear un nuevo proyecto para la gestión y monitorización de desarrollo en la plataforma ***CyScrum***.
2. Agregar colaboradores al proyecto creado, permitiendo trabajar colaborativamente.
3. Comprobar el acceso a los colaboradores en el proyecto creado.
4. Comprobar el registro del proyecto creado y los colaboradores agregados en la base de datos de la plataforma ***CyScrum***.

Procedimiento

1. Abrir el navegador web FireFox Mozilla.
2. Dirigirse a https://localhost:44367/.
3. Iniciar sesión con el usuario **José Raúl**.
4. Crear un proyecto en la plataforma ***CyScrum*** (**HU 3**).
5. Verificar en la base de datos el registro del proyecto creado en la plataforma ***CyScrum***.
6. Agregar como colaborador al usuario **José Alberto** en el proyecto creado (**HU 4**).

7. Verificar en la base de datos el registro del colaborador en el proyecto creado.
8. Iniciar sesión con el usuario **José Alberto** y seleccionar el proyecto creado.
9. Comparar las restricciones entre los usuarios **José Raúl** y **José Alberto** en la plataforma ***CyScrum***.

Desarrollo

1. Abrir el navegador web FireFox Mozilla.

En la computadora con sistema operativo Windows 10 se procede abrir el navegador web FireFox Mozilla, dando clic en el icono de aplicación (ver figura 5.16).

Figura 5.16. Icono del navegador web FireFox Mozilla.

2. Dirigirse a https://localhost:44367/.

En la barra de navegación del navegador web FireFox se escribe la dirección https://localhost:44367 y se presiona la tecla [ENTER] (ver figura 5.17).

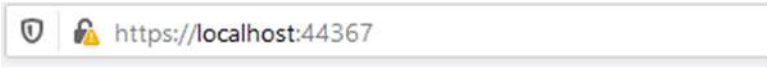


Figura 5.17. Barra de navegación de FireFox.

3. Iniciar sesión con el usuario **José Raúl**.

Para crear un proyecto en la plataforma ***CyScrum*** es necesario iniciar sesión. Como se observa en la figura 5.18, se introducen las credenciales del usuario **José Raúl** previamente creado y posteriormente se selecciona el botón **Iniciar sesión** (ver indicador 1) para iniciar sesión.

Figura 5.18. El usuario José Raúl introduciendo sus credenciales.

Figura 5.19. Formulario para seleccionar un proyecto existente.

Cuando se accede con el usuario **José Raúl** a la plataforma ***CyScrum*** se muestra un formulario para poder seleccionar un proyecto a través de una caja de selección (*combo box*) con etiqueta **Selecciona un proyecto** (ver indicador 1 de la figura 5.19), pero en caso de no

contar con un proyecto existente, para crear uno se tiene que hacer clic en el hipervínculo **Nuevo proyecto** (ver indicador 2).

4. Crear un proyecto en la plataforma ***CyScrum*** (**HU 3**).

Cuando se hace clic en el hipervínculo **Nuevo proyecto** (ver figura 5.19), se muestra un formulario emergente (*modal pop-up*) (ver indicador 1 en la figura 5.20) para realizar el registro de un nuevo proyecto, como se muestra en la figura 5.20.

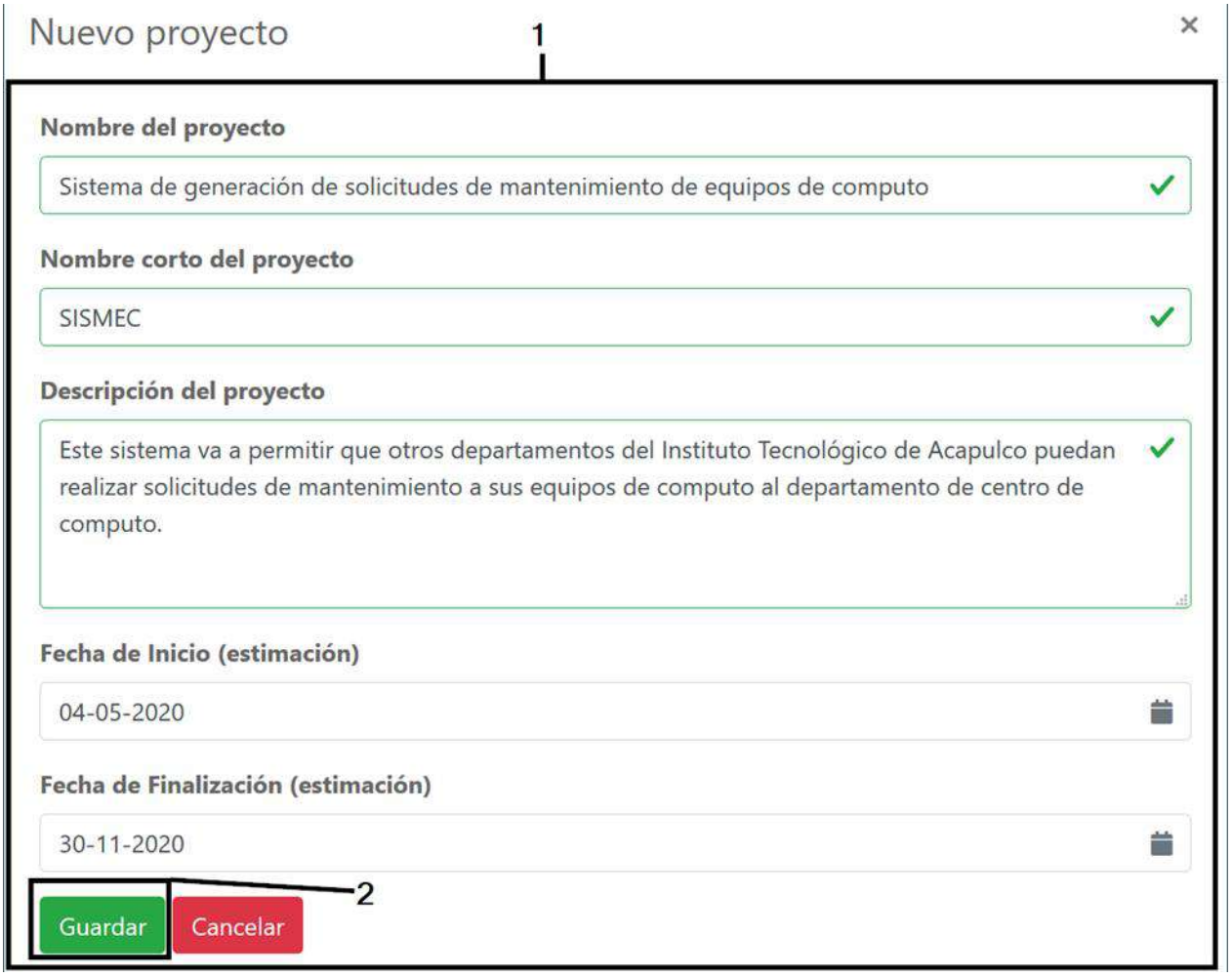


Figura 5.20. Formulario pata crear un nuevo proyecto.

Una vez que se haya proporcionado los datos en el formulario **Nuevo proyecto**, posteriormente para crear el proyecto con los datos proporcionados es necesario seleccionar el botón **Guardar** (ver indicador 2 de la figura 5.20). Al seleccionar el botón **Guardar** se

envían los datos al servidor y son almacenados en la base de datos que está contenida en el gestor de base de datos **PostgreSQL**.

5. Verificar en la base de datos el registro del proyecto creado en la plataforma ***CyScrum***.

Para verificar que los datos introducidos en el formulario **Nuevo Proyecto** se guardaron correctamente, se realizó una consulta a la tabla ***Proyectos*** de la base de datos de la plataforma ***CyScrum***. Como se observa en la figura 5.21, se refleja el registro en la base de datos del nuevo proyecto llamado con nombre corto **SISMEC** (ver figura 5.20) que se registró en el formulario **Nuevo Proyecto**.

	IdProyecto [PK] numeric(20,0)	NombreProyecto character varying(300)	NombreCortoProyecto character varying(25)
1	504052020095413	Sistema de generación de solicitudes de mantenimiento de equipos de comput	SISMEC

Figura 5.21. Consulta del registro del nuevo proyecto SISMEC.

6. Agregar como colaborador al usuario **José Alberto** en el proyecto creado (**HU 4**).

Cuando se creó el proyecto **SISMEC**, la plataforma ***CyScrum*** se re direcciona a la vista que se muestra en la figura 5.22.

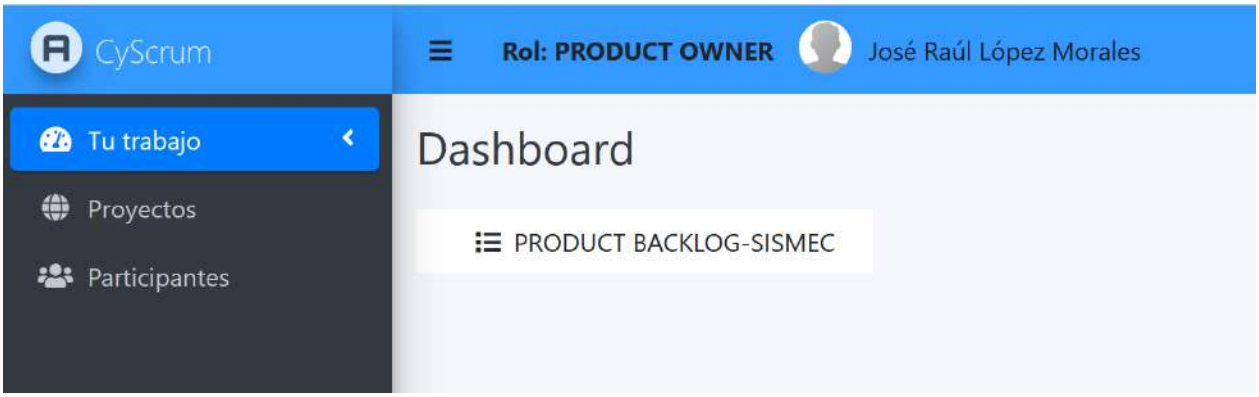


Figura 5.22. Dashboard de la plataforma CyScrum.

Esta vista es el área de trabajo donde el usuario **José Raúl** puede gestionar las historias de usuario, Sprints y tareas del proyecto **SISMEC**.

Para agregar colaboradores al proyecto **SISMEC**, se selecciona la opción **Participantes** (α) que se encuentra en el menú de la parte lateral izquierda. Al hacer esta acción la plataforma ***CyScrum*** se re direcciona a la vista que se muestra en la figura 5.23.

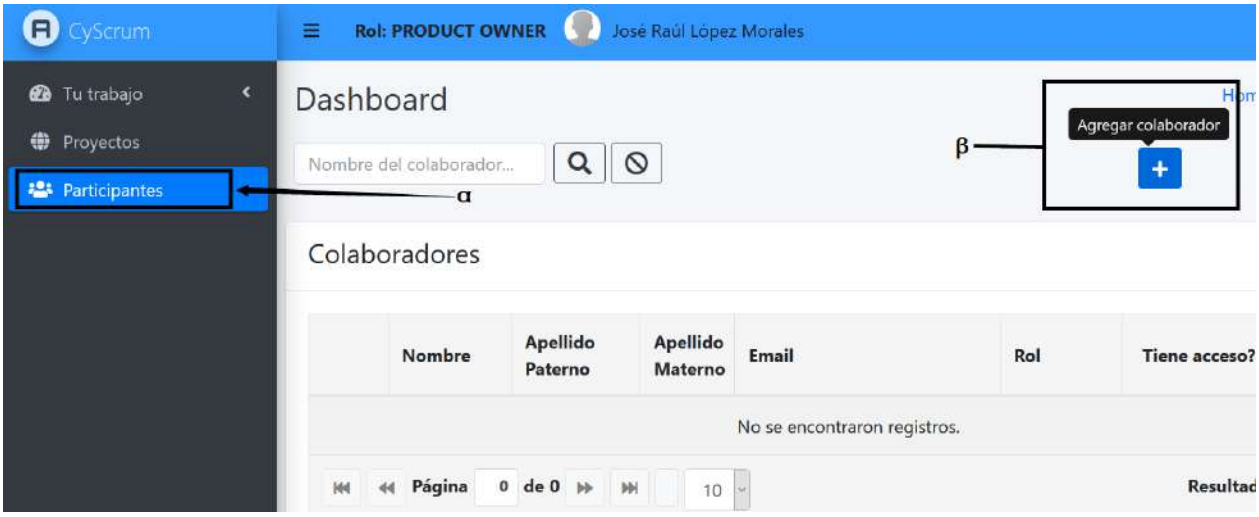


Figura 5.23. Vista para mostrar la lista de colaboradores en un proyecto.

Cuando se crea un proyecto, éste no cuenta con colaboradores como se muestra en la tabla de la vista **Colaboradores** (ver figura 5.23). Para agregar colaboradores al proyecto **SISMEC** se selecciona el botón con leyenda **Agregar colaborador** (β), entonces se muestra un *modal-popup* con el formulario para agregar un colaborador a un proyecto como se muestra en la figura 5.24

Figura 5.24. Formulario para agregar un colaborador al proyecto.

En el formulario **Nuevo Colaborador** (ver figura 5.24) se proporciona una dirección de correo electrónico de un usuario registrado en la plataforma, el rol que tomará el colaborador en el proyecto y una tarifa por hora que invierta en las tareas asignadas, con estos datos proporcionados, posteriormente el usuario **José Raúl** selecciona el botón **Guardar** (β). Cuando el usuario selecciona el botón **Guardar** automáticamente el formulario **Nuevo Colaborador** desaparece y la tabla de la lista de los colaboradores se actualiza y se ve reflejado en el registro del colaborador agregado (ver figura 5.25).

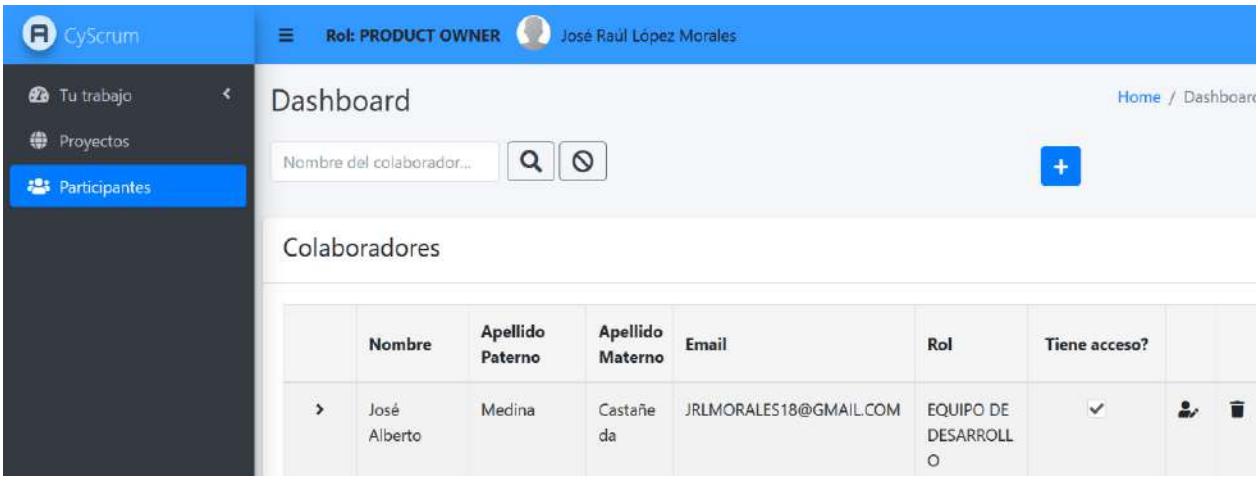


Figura 5.25. Vista Colaboradores con un colaborador agregado en el proyecto SISMEC.

7. Verificar en la base de datos el registro del colaborador en el proyecto creado.

NombreCortoProyecto character varying(25)	Nombre character varying(70)	TarifaXHora integer	NombreRol character varying(80)	TieneAcceso boolean
SISMEC	José Alberto	100	EQUIPO DE DESARRO	t

Figura 5.26. Consulta del registro del colaborador en el proyecto SISMEC.

En la figura 5.26 se observa el registro del colaborador (usuario **José Alberto**) que fue agregado al proyecto **SISMEC**. Para restringir o proporcionar acceso a un colaborador en un proyecto, se controla con el campo **TieneAcceso**, en este caso el usuario **José Alberto** tiene acceso al proyecto porque en el campo mencionado tiene el valor de **true** indicado con una **t.**

8. Iniciar sesión con el usuario **José Alberto** y seleccionar el proyecto creado.

Cuando un usuario accede a la plataforma, lo primero es seleccionar un proyecto, para ello se ingresó a la plataforma ***CyScrum*** con el usuario **José Alberto Medina Castañeda** previamente ingresado, para comprobar si tiene acceso al proyecto **SISMEC**. En caso de tener acceso, el proyecto **SISMEC** aparecerá en la lista proporcionada por el *combo box* (α) (ver figura 5.27)

Figura 5.27.Vista para seleccionar un proyecto con un usuario diferente.

9. Comparar las restricciones entre los usuarios **José Raúl** y **José Alberto** en la plataforma ***CyScrum***.

Con la sesión del usuario **José Alberto** selecciona el proyecto y hace clic en el botón **Seleccionar** (β), la plataforma lo re direccionará a la vista que es muy similar a la se muestra en la figura 5.22. La diferencia entre el usuario **José Raúl** que creó el proyecto **SISMEC** (rol de product owner) y el usuario **José Alberto** que tiene el rol de equipo de desarrollo (colaborador), radica en las opciones que contiene el menú que está ubicado en la parta lateral

izquierda como se observa en las figuras 5.28 y 5.29. E l tipo de rol establece donde un usuario puede navegar, además permite restringir las operaciones que puede realizar en un proyecto.

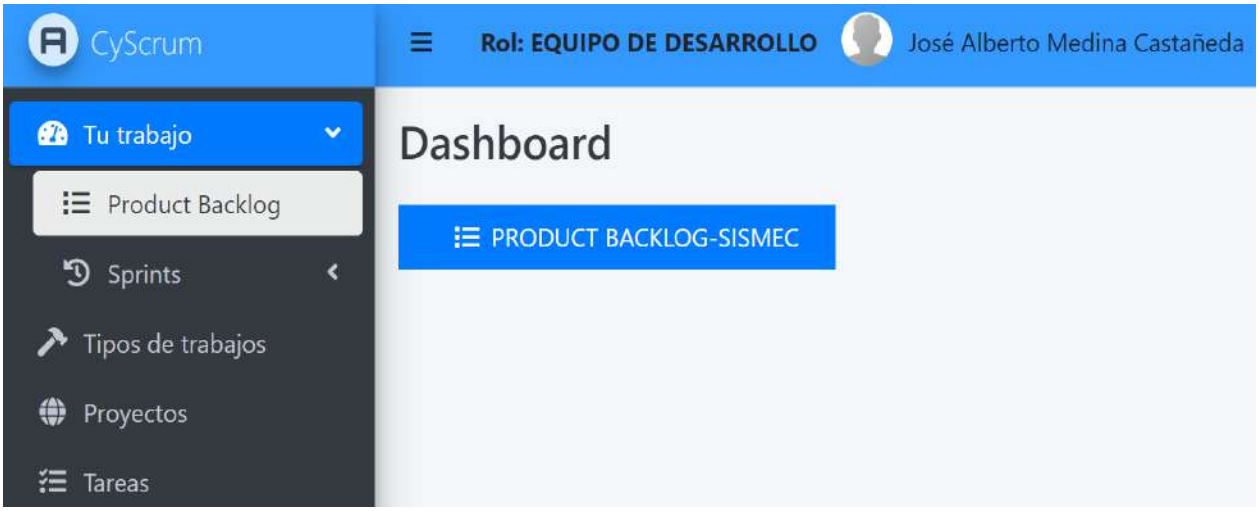


Figura 5.28. Vista principal de la plataforma con un usuario de tipo rol equipo de desarrollo.

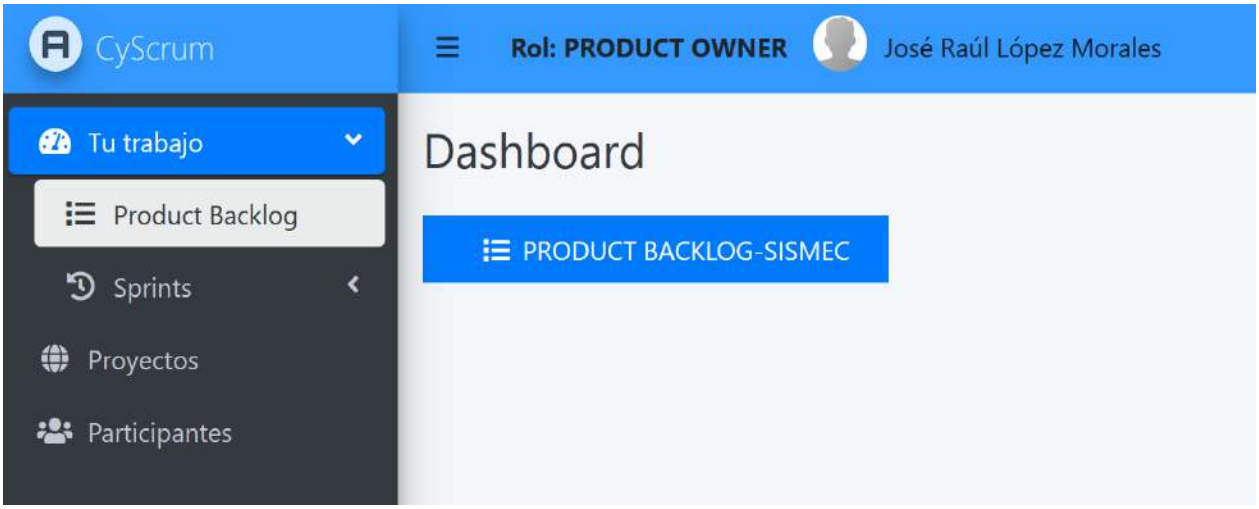


Figura 5.29. Vista principal de la plataforma con un usuario de tipo rol product owner.

Conclusión

Al verificar el correcto funcionamiento para crear nuevos proyectos y agregar colaboradores a ellos en conjunto con las vistas correspondientes. Con esto no solo se demuestra que se han cumplido con los objetivos establecidos en este caso de estudio sino también la plataforma cumple con los criterios que establece la metodología Scrum en los

roles establecidos por la misma. Además de las **HU 3** y **HU 4** que se desarrollaron en el Sprint 2.

5.4 Caso de estudio 3: Gestión de historias de usuario

Descripción

En el presente caso de estudio se validará el funcionamiento para agregar y priorizar las historias de usuario en el product backlog del proyecto **SISMEC** previamente creado y la creación de los Sprints, además de la presentación de las historias de usuario en los Sprints. Con el objetivo de comprobar con las **HU 5** a **HU 10**.

Objetivos

1. Agregar una nueva historia de usuario al product backlog del proyecto **SISMEC**.
2. Priorizar las historias de usuario del product backlog.
3. Crear un nuevo Sprint en el proyecto **SISMEC**.
4. Mover historias de usuario del product backlog al Sprint creado.
5. Verificar en la base de datos si se guardó correctamente la de historia de usuario y Sprint.

Procedimiento

1. Abrir el navegador web FireFox Mozilla.
2. Dirigirse a https://localhost:44367/.
3. Iniciar sesión con el usuario **José Raúl**.
4. Seleccionar el proyecto **SISMEC** que fue creado previamente.
5. Agregar una historia de usuario al product backlog del proyecto **SISMEC** (**HU 5 y 8**).
6. Verificar en la base de datos que la historia de usuario que se agregó, se guardó correctamente.

7. Priorizar una historia de usuario del product backlog del proyecto **SISMEC** (**HU 7**).
8. Verificar que en la base de datos el orden de la historia de usuario priorizada se actualizó correctamente.
9. Crear un Sprint en el proyecto **SISMEC** (**HU 6 y 10**).
10. Verificar en la base de datos que el Sprint creado en el proyecto **SISMEC**, se guardó correctamente.
11. Mover una historia de usuario del product backlog al Sprint creado en el proyecto **SISMEC** (**HU 9**).
12. Verificar en la base de datos si la información de la historia de usuario se actualizó correctamente, al ser movida en a un Sprint.

Desarrollo.

1. Abrir el navegador web FireFox Mozilla.

   En la computadora con sistema operativo Windows 10 se procede abrir el navegador web FireFox Mozilla, dando clic en el icono de aplicación (ver figura 5.30).

Figura 5.30. Icono del navegador web FireFox Mozilla

2. Dirigirse a https://localhost:44367/.

   En la barra de navegación del navegador web FireFox se escribe la dirección https://localhost:44367 y se presiona la tecla [ENTER] (ver figura 5.31).

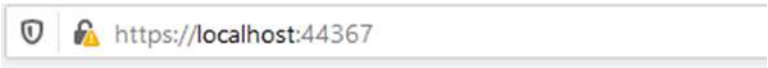


Figura 5.31. Barra de navegación de FireFox.

3. Iniciar sesión con el usuario **José Raúl**.

Para el registro de historias de usuario en un proyecto creado en la plataforma ***CyScrum***, es necesario iniciar sesión. Como se observa en la figura 5.32, se introducen las credenciales del usuario **José Raúl** previamente creado y posteriormente se selecciona el botón **Iniciar sesión** (ver indicador 1) para iniciar sesión.

Figura 5.32. El usuario José Raúl introduciendo sus credenciales.

4. Seleccionar el proyecto **SISMEC** que fue creado previamente.

Cuando se accede con el usuario **José Raúl** a la plataforma ***CyScrum***, se muestra un formulario para seleccionar un proyecto, donde se eligió el proyecto **SISMEC** previamente creado (ver indicador 1 en la figura 5.33). Posteriormente de haber elegido el proyecto **SISMEC**, para poder gestionarlo y monitorizarlo es necesario seleccionar el botón **Seleccionar** (ver indicador 2).

Figura 5.33. Formulario para seleccionar el proyecto SISMEC.

5. Agregar una historia de usuario al product backlog del proyecto **SISMEC** (**HU 5 y 8**).

Después de haber seleccionado el botón **Seleccionar**, la plataforma se re direccionará a la vista que se muestra en la figura 5.34.

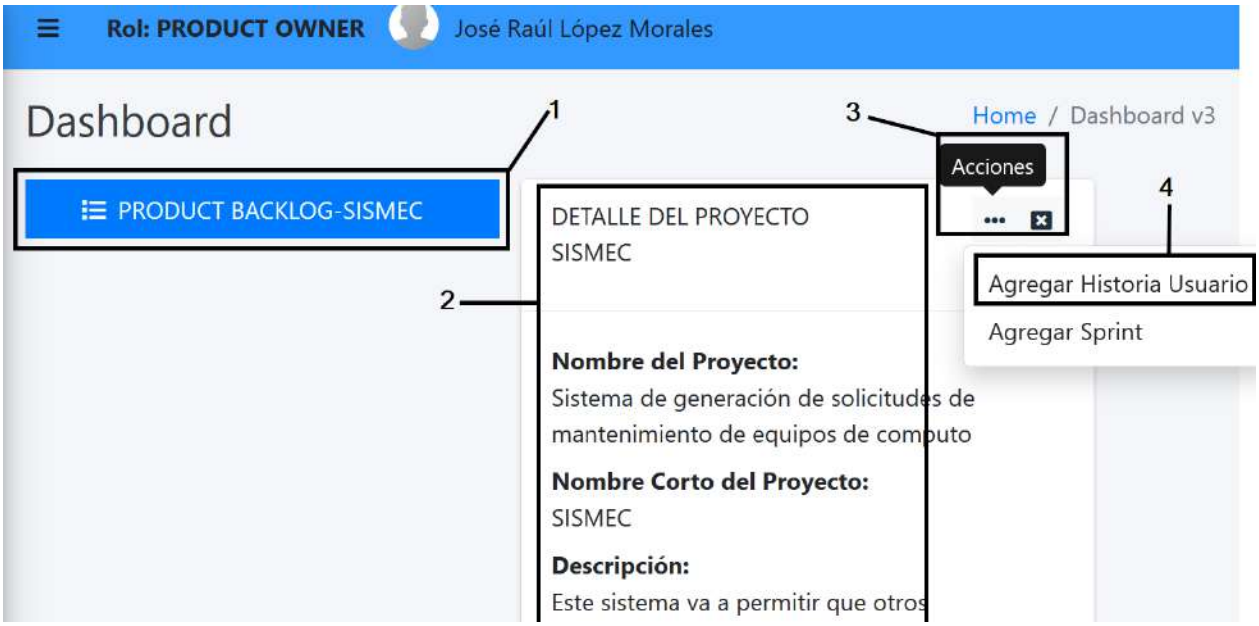


Figura 5.34. Vista principal de la plataforma para gestionar un proyecto.

Para agregar **historias de usuario** al **product backlog** es necesario hacer clic sobre él (ver indicador 1) y aparecerá a la derecha un recuadro con la información general del proyecto (ver indicador 2). En parte superior del recuadro se encuentra un botón con tres puntos

posicionados horizontalmente, el cual tiene una leyenda **Acciones** (ver indicador 3), al hacer clic se muestra un menú de opciones. Para agregar una nueva historia de usuario se tiene que hacer clic en la opción **Agregar Historia Usuario** (ver indicador 4).

Cuando se haya hecho clic en la opción **Agregar Historia Usuario** aparecerá un *modal pop-up* con un formulario llamado **Historia de usuario** para introducir información para la nueva historia de usuario como se observa en la figura 5.35.

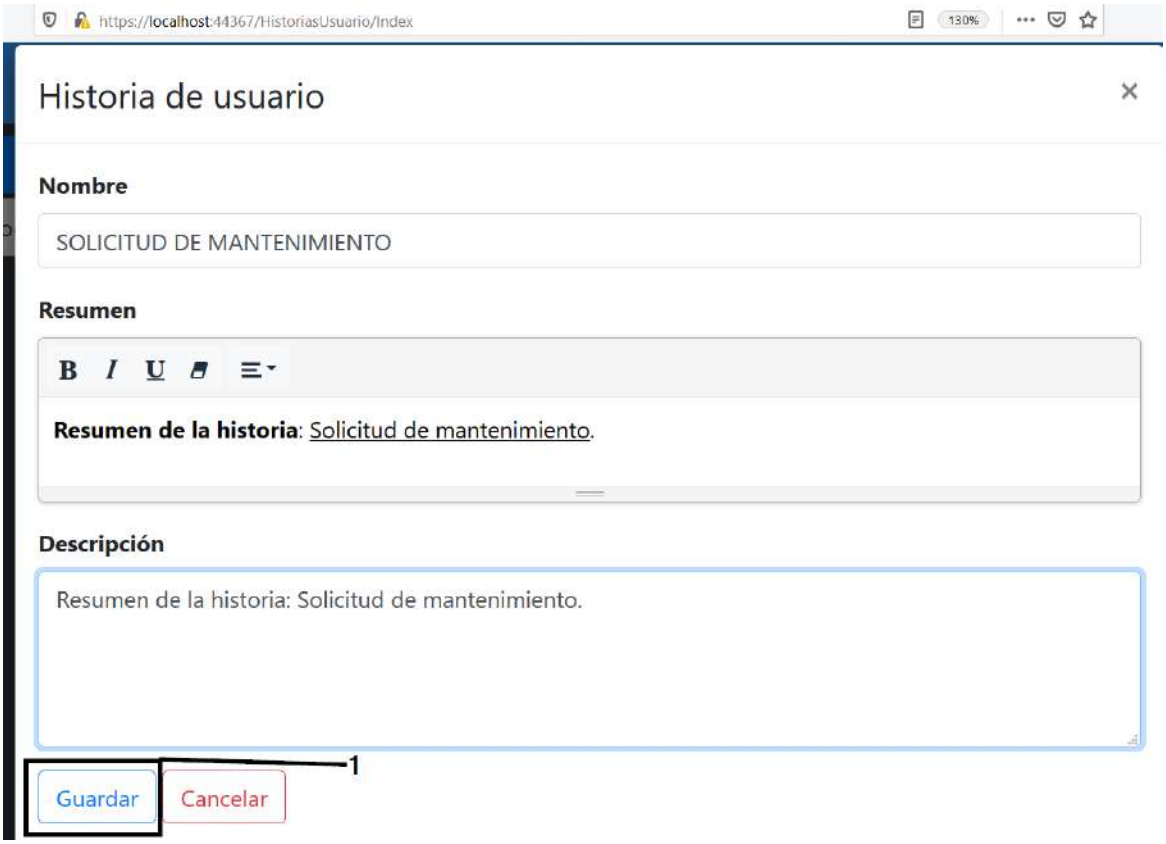


Figura 5.35. Formulario para introducir los datos para una nueva historia de usuario.

Al presionar el botón **Gurdar** (ver indicador 1) se envían los datos al servidor y se almacenan en la base de datos, y si todo se realiza correctamente aparecerá un mensaje de éxito, como se observa en la figura 5.36.

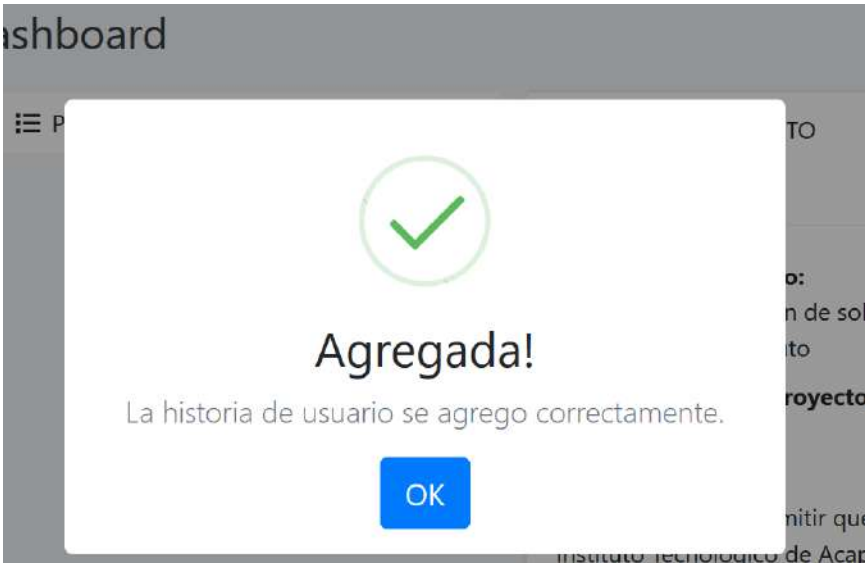


Figura 5.36. Mensaje de éxito cuando se agrega una historia de usuario.

Después de haberse agregado la historia de usuario correctamente, el product backlog se actualiza automáticamente y se ve reflejada la historia de usuario **SOLICITUD DE MANTENIMIENTO** (ver indicador 1 en la figura 5.37).

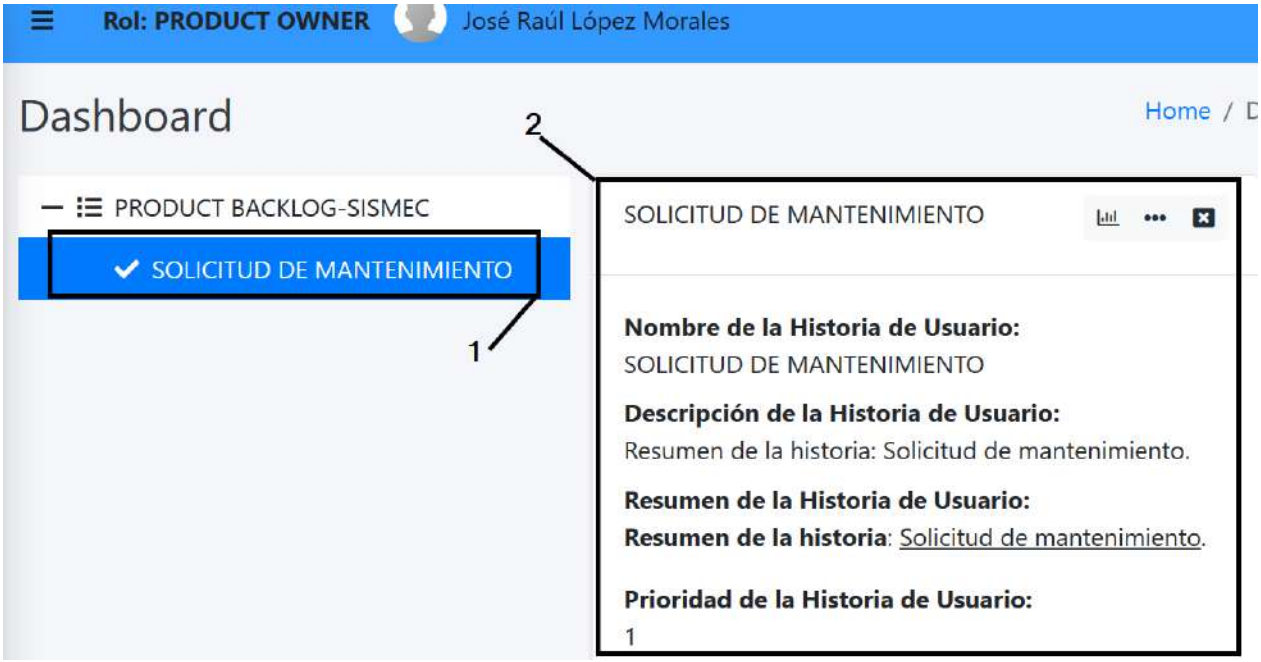


Figura 5.37. Historia de usuario mostrada en el product backlog.

De la misma forma para consultar información del product backlog, al hacer clic sobre la historia de usuario **SOLICITUD DE MANTENIMIENTO** aparece un recuadro con la información de esta (ver indicador 2).

6. Verificar en la base de datos que la historia de usuario que se agregó, se guardó correctamente.

Con la creación de la historia de usuario **SOLICITUD DE MANTENIMIENTO** (ver indicador 1) en la plataforma ***CyScrum***, se puede observar en la figura 5.38 que se guardó correctamente en la base de datos. Además, se agregaron dos historias de usuarios (**INVENTARIO DE EQUIPOS** y **GESTIÓN DE USUARIOS**) (ver indicador 2).

NombreHistoriaUsuario character varying(70)	NumOrden integer	NombreCortoProyecto character varying(25)	
SOLICITUD DE MANTENIMIENTO	1	SISMEC	1
INVENTARIO DE EQUIPOS	2	SISMEC	2
GESTIÓN DE USUARIOS	3	SISMEC	

Figura 5.38. Verificación del registro de la historia de usuario SOLICITUD DE MANTENIMIENTO.

7. Priorizar una historia de usuario del product backlog del proyecto **SISMEC** (**HU 7**).

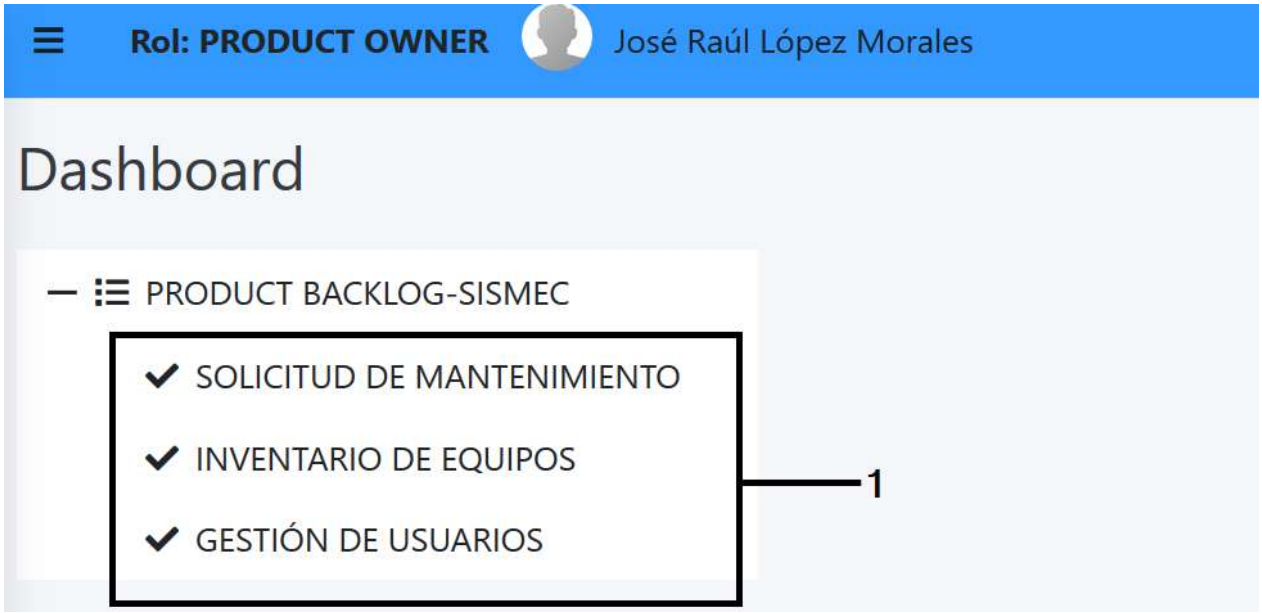


Figura 5.39. Historias de usuario contenidas en el product backlog.

Para priorizar o cambiar el orden de una historia de usuario en el product backlog basta con que se arrastre y se suelte en la posición deseada, en la figura 5.39 se muestran las tres

historias de usuario que están almacenas en la base de datos. El orden de las historias de usuario está almacenado en el campo **NumOrden** , como se observa en la figura 5.38, donde la historia de usuario **SOLICITUD DE MANTENIMIENTO** tiene mayor prioridad que las demás por tener el valor de 1.

Cuando se prioriza una historia de usuario, cambia su número de prioridad, como se observa en la figura 5.40 al priorizar la historia de usuario **SOLICITUD DE MANTENIMIENTO** (α) su número de prioridad cambia y esto se puede ver al consultar la información de la historia de usuario (β), donde tenía en un inicio el valor de 1, posteriormente a la priorización su valor cambio a 2.

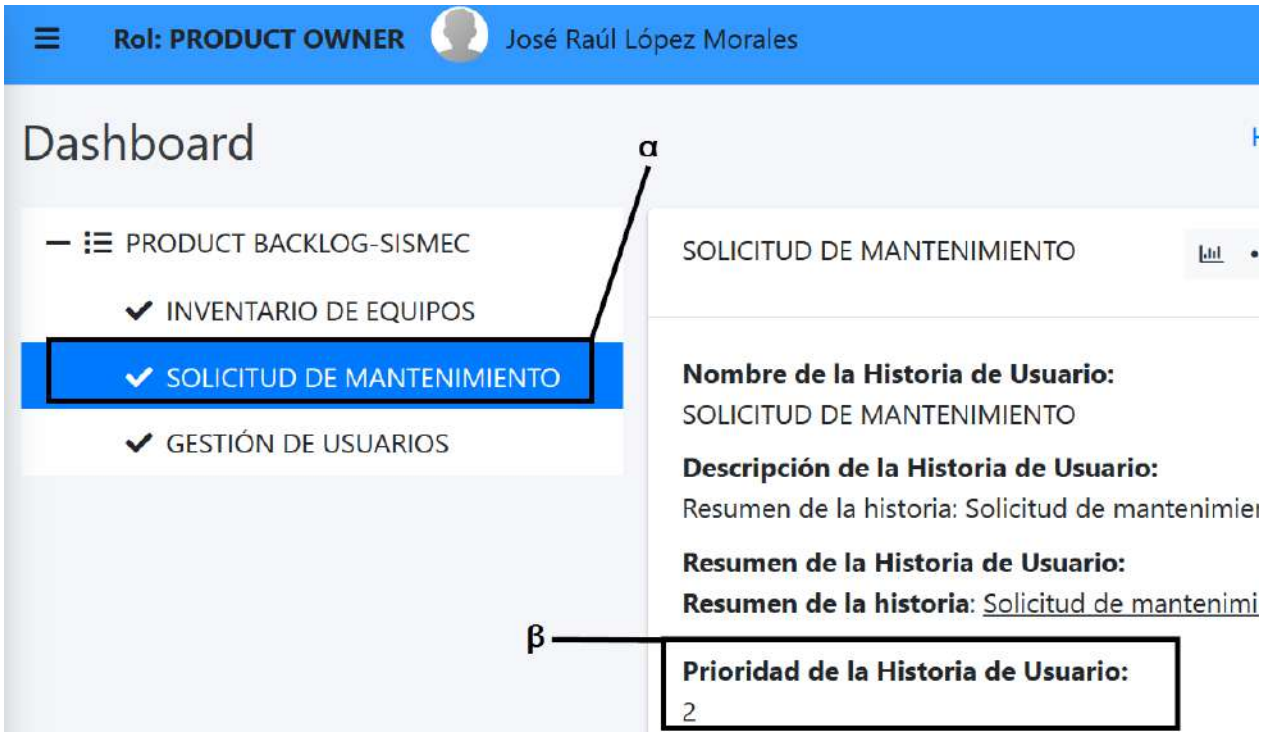


Figura 5.40. Cambio de prioridad a la historia de usuario SOLICITUD DE MANTENIMIENTO.

8. Verificar que en la base de datos el orden de la historia de usuario priorizada se actualizó correctamente.

Para comprobar que la información de la historia de usuario **SOLICITUD DE MATENIMIENTO** se actualizó correctamente después de haber cambiado su prioridad, se realizó una consulta en la base de datos, donde se puede observar en el campo **NumOrden** del registro que tiene la historia de usuario **SOLICITUD DE MANTENIMIENTO** cambio su valor de 1 a 2 (ver indicador 1 en la figura 5.41).

NombreHistoriaUsuario character varying(70)	NumOrden integer	NombreCortoProyecto character varying(25)
INVENTARIO DE EQUIPOS	1	SISMEC
SOLICITUD DE MANTENIMIENTO	2	SISMEC
GESTIÓN DE USUARIOS	3	SISMEC

1

Figura 5.41. Registros de las historias de usuario después de haber cambiado el orden.

9. Crear un Sprint en el proyecto **SISMEC** (**HU 6 y 10**).

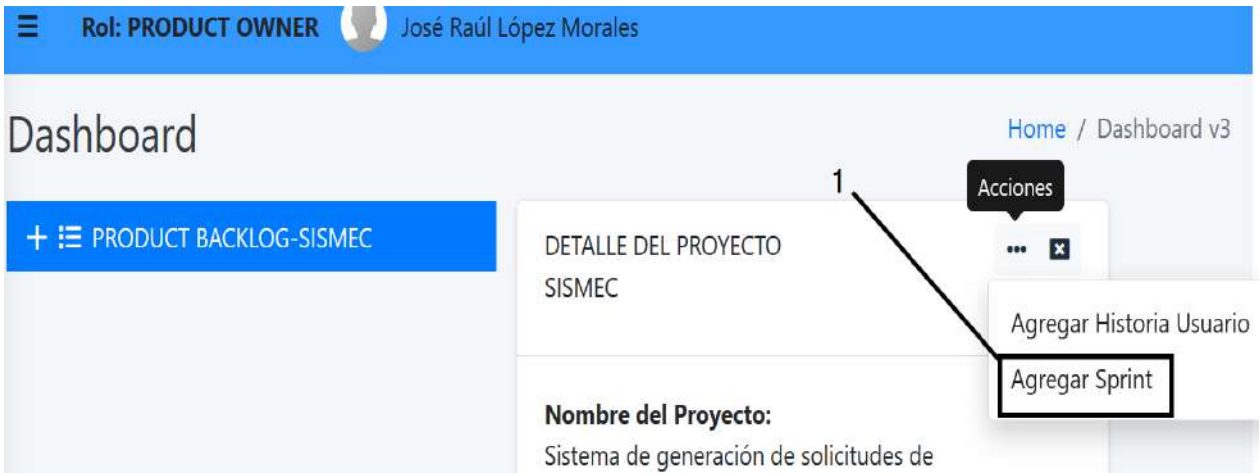


Figura 5.42. Vista del área de trabajo de la plataforma CyScrum.

Para presentar historias de usuario en un Sprint, primero es necesario crear uno, para ello el proceso es muy similar al de crear una historia de usuario con la única excepción que se tiene que hacer clic en la opción **Agregar Sprint** (ver indicador 1 en la figura 5.42). Al momento de hacer clic en la opción **Agregar Sprint**, muestra un *modal pop-up* que contiene un formulario para introducir datos para la creación de un Sprint (ver figura 4.43)

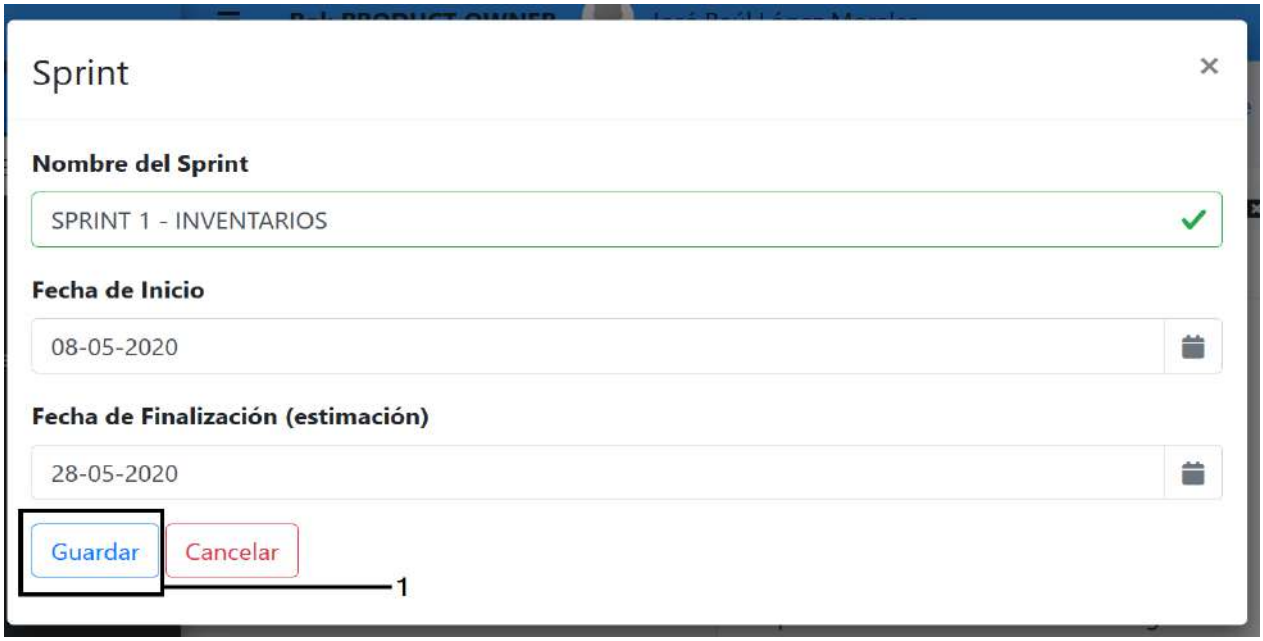


Figura 5.43. Formulario para crear un Sprint en un proyecto.

Una vez proporcionado los datos para crear un Sprint se selecciona el botón **Guardar** (ver indicador 1 en la figura 5.43), si los datos son correctos entonces la plataforma ***CyScrum*** envía un mensaje de éxito como se muestra en la figura 5.44.

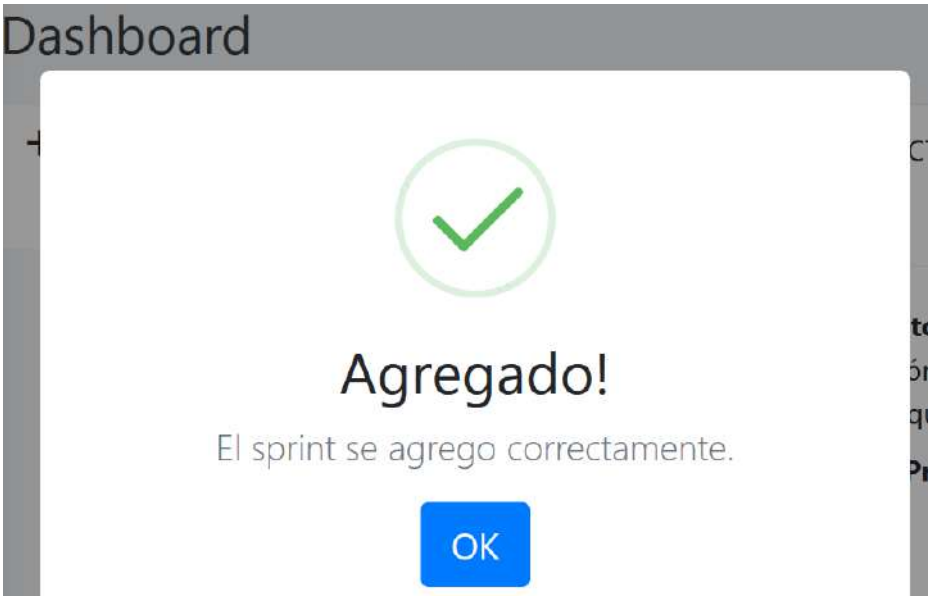


Figura 5.44. Mensaje de éxito después de haber agregado un Sprint.

10. Verificar en la base de datos que el Sprint creado en el proyecto **SISMEC**, se guardó correctamente.

Se realizó una consulta en la tabla Sprint de la base de datos de la plataforma ***CyScrum*** para verificar que el Sprint con nombre **SPRINT 1 - INVENTARIOS** previamente creado, se haya guardado correctamente. En la figura 5.45 se puede observar el registro del Sprint **SPRINT 1 – INVENTARIOS** con id **620605202001425**.

IdSprint [PK] numeric(20,0)	NombreSprint character varying(150)	FechaApertura timestamp with	FechaTerminacion timestamp without	Estimacion integer	Estatus character va
620605202001425	SPRINT 1 - INVENTARIOS	2020-05-08 00	2020-05-28 00:00	0	PENDIENTE

Figura 5.45. Registro de un Sprint en la base de datos.

11. Mover una historia de usuario del product backlog al Sprint creado en el proyecto **SISMEC** (**HU 9**).

Para mover historias de usuario del **product backlog** a un **Sprint**, el proceso es similar al priorizar historias de usuario, la diferencia es que son deslizadas al Sprint deseado. La tabla que contiene las **historias de usuario** está relacionada con la tabla que contiene los registros de los **Sprints**, en la figura 5.46 los registros de las **historias de usuario** en el campo **IdSprint** no tienen un valor (ver indicador 1), esto permite simular que las historias de usuario aún están contenidas en el **product backlog**.

NombreHistoriaUsuario character varying(70)	NumOrden integer	NombreCortoProyecto character varying(25)	IdSprint numeric(20,0)
INVENTARIO DE EQUIPOS	1	SISMEC	
SOLICITUD DE MANTENIMIENTO	2	SISMEC	
GESTIÓN DE USUARIOS	3	SISMEC	

1

Figura 5.46. Consulta de los registros de las historias de usuario de un proyecto.

En la figura 5.47 se puede observar que la historia de usuario **SOLICITUD DE MANTENIMIENTO** fue movida en el Sprint **SPRINT 1 – INVENTARIOS** (ver indicador 1), al consultar la información de la historia de usuario **SOLICITUD DE**

MANTENIMIENTO se puede observar que ahora en la etiqueta **Sprint** tiene el nombre del Sprint donde está contenida (ver indicador 2).

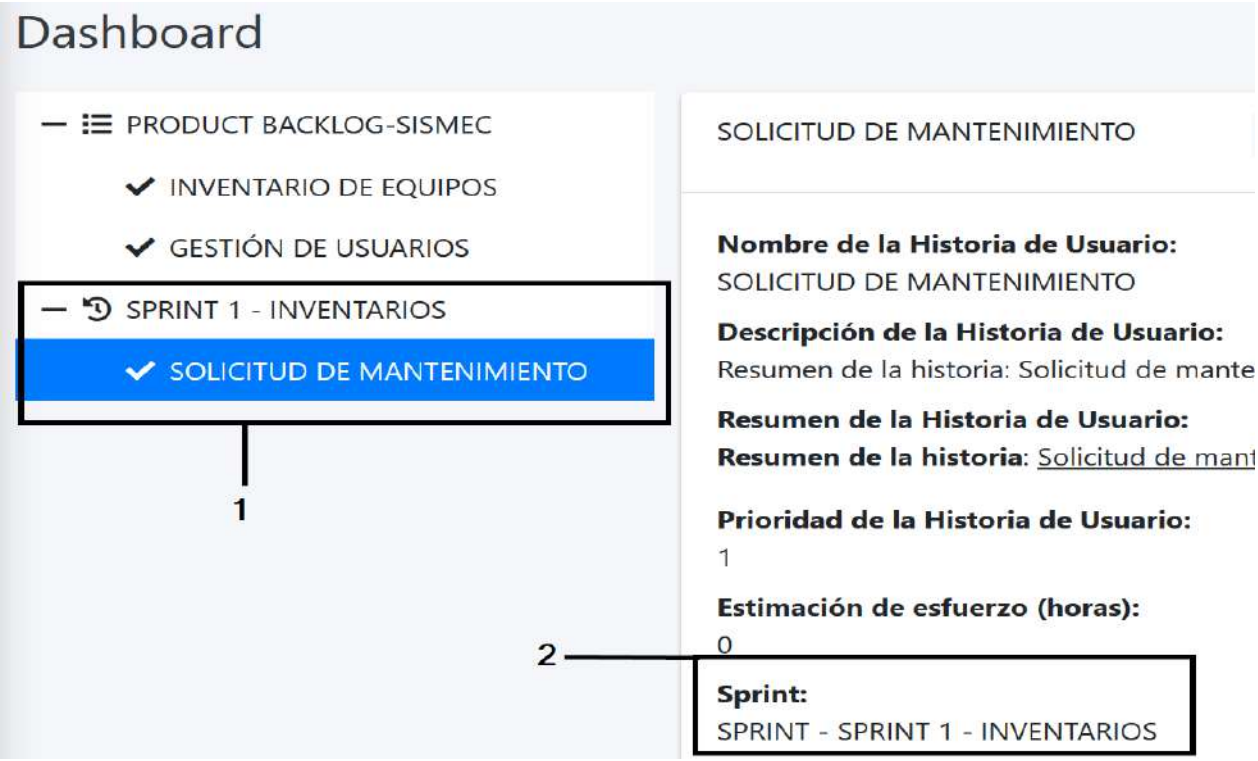


Figura 5.47. Historia de usuario contenida en un Sprint.

12. Verificar en la base de datos si la información de la historia de usuario se actualizó correctamente, al ser movida en a un Sprint.

NombreHistoriaUsuario character varying(70)	NumOrden integer	NombreCortoProyecto character varying(25)	IdSprint numeric(20,0)
SOLICITUD DE MANTENIMIENTO	1	SISMEC	620605202001425
INVENTARIO DE EQUIPOS	1	SISMEC	
GESTIÓN DE USUARIOS	2	SISMEC	

1

Figura 5.48. Registro de las historias de usuario en la base de datos cuando está contenida en un Sprint.

Al realizar una consulta a la tabla ***HistoriasUsuario*** en la base de datos, la historia de usuario **SOLICITUD DE MANTENIMIENTO** en el campo **IdSprint** tiene como valor el id del Sprint **SPRINT 1 – INVENTARIOS** (**620605202001425**) (ver indicador 1), como se observa en la figura 5.48.

Conclusión

Con respecto a los resultados que se presentaron en este caso de estudio, se cumplieron con los objetivos establecidos al inicio de este, con la creación historias de usuario y Sprint, ademas de la priorización de las historias de usuario en el product backlog. De esta forma se cumplen las HU 5 a HU 10 que fueron implementadas en el Sprint 3 presentado en el capítulo 4.

5.5 Caso de estudio 4: División de las historias de usuario en tareas y creación de tipos de trabajo

Descripción

En este caso de estudio se validará el funcionamiento para la creación de tareas en las historias de usuario contenidas en un Sprint, así como la creación de los tipos de trabajos que integrante del equipo de desarrollo realizará en una tarea asignada.

Objetivos

1. Agregar una tarea a una historia de usuario contenida en un Sprint.
2. Asignar un integrante del equipo de desarrollo a la tarea creada.
3. Crear un tipo de trabajo para una tarea en el catálogo de tipos de trabajos.
4. Verificar en la base de datos si se guardó correctamente una tarea.
5. Verificar en la base de datos si se actualizó la tarea al ser asignada.

Procedimiento

1. Abrir el navegador web FireFox Mozilla.
2. Dirigirse a https://localhost:44367/.
3. Iniciar sesión con el usuario **José Alberto**.
4. Seleccionar el proyecto **SISMEC** que fue creado previamente.

5. Agregar una tarea a la historia de usuario **SOLICITUD DE MANTENIMIENTO** (**HU 11 ,12 Y 13**).
6. Verificar en la base de datos el registro de la tarea creada.
7. Asignar la tarea creada al usuario **José Alberto** (**HU 15**).
8. Verificar en la base de datos que la información de la tarea se actualizó al momento de asignarle un integrante del equipo de desarrollo.
9. Crear un tipo de trabajo en el catálogo de tipos de trabajos del proyecto **SISMEC** (**HU 16**).
10. Verificar en la base de datos que el tipo de trabajo creado se guardó correctamente.

Desarrollo

1. Abrir el navegador web FireFox Mozilla.

En la computadora con sistema operativo Windows 10 se procede abrir el navegador web FireFox Mozilla, dando clic en el icono de aplicación (ver figura 5.49).

Figura 5.49. Icono del navegador web FireFox Mozilla.

2. Dirigirse a https://localhost:44367/.

En la barra de navegación del navegador web FireFox se escribe la dirección https://localhost:44367 y se presiona la tecla [ENTER] (ver figura 5.50).

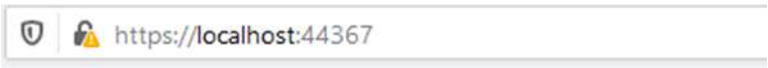


Figura 5.50. Barra de navegación de FireFox.

3. Iniciar sesión con el usuario **José Alberto**.

Para creación de tareas en la plataforma ***CyScrum***, es necesario acceder a un proyecto con rol de equipo de desarrollo, pero primero se necesita iniciar sesión. Como se observa en la figura 5.51, se introducen las credenciales del usuario **José Alberto** previamente creado y posteriormente se selecciona el botón **Iniciar sesión** (ver indicador 1).

Figura 5.51.Vista para iniciar sesión en la plataforma CyScrum.

4. Seleccionar el proyecto **SISMEC** que fue creado previamente.

Como se mencionó anteriormente es necesario acceder a un proyecto con rol de equipo de desarrollo establecido, para agregar tareas a las historias de usuario. Por tal motivo se elige el proyecto **SISMEC** (α), en el cual el usuario **José Alberto** fue invitado como colaborador con rol de equipo de desarrollo. Posteriormente de haber elegido el proyecto se selecciona el botón **Seleccionar** (β) (ver figura 5.52).

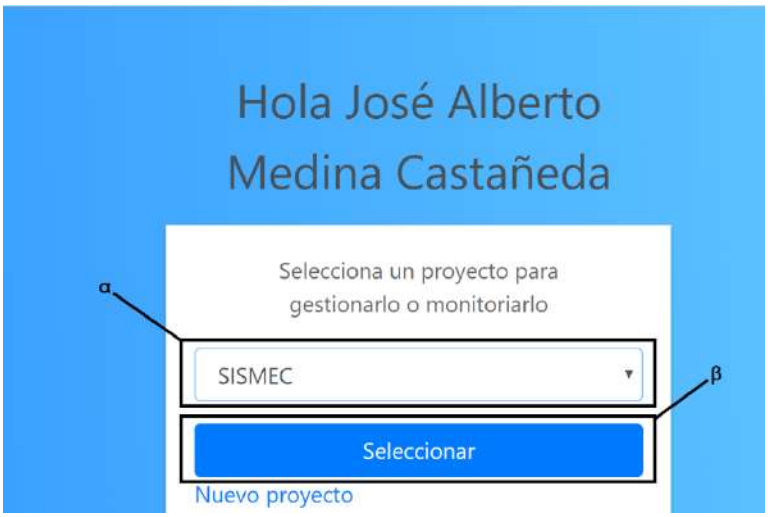


Figura 5.52. Vista para seleccionar un proyecto en la plataforma CyScrum.

Después de haber seleccionado el proyeto **SISMEC**, se muestra el **product backlog** del proyecto y un sprint (**SPRINT 1 - INVENTARIOS**) (ver indicador 2), donde el **product backlog** cuenta con una historia de usuario (**INVENTARIO DE EQUIPOS**) (ver indicador 1) y el sprint con dos historias de usuario (**SOLICITUD DE MANTENIMIENTO y GESTIÓN DE USUARIOS**) (ver indicador 2) como se observa en la figura 5.53.

Figura 5.53. Product backlog y Sprint del proyecto SISMEC.

5. Agregar una tarea a la historia de usuario **SOLICITUD DE MANTENIMIENTO** (**HU 11 ,12 Y 13**).

Para agregar una tarea a una historia de usuario se hace clic en una de las historias de usuario contenidas en el sprint (**SOLICITUD DE MANTENIMIENTO**) (ver indicador 1), al realizar esto se muestra un recuadro con la información de la historia de usuario en la parte derecha (ver indicador 2 en la figura 5.54). En la parte superior del recuadro tiene un icono con tres puntos horizontalmente con leyenda **Acciones** (ver indicador 3), al dar clic sobre el aparece un menú con la opción **Agregar tarea** (ver indicador 4).

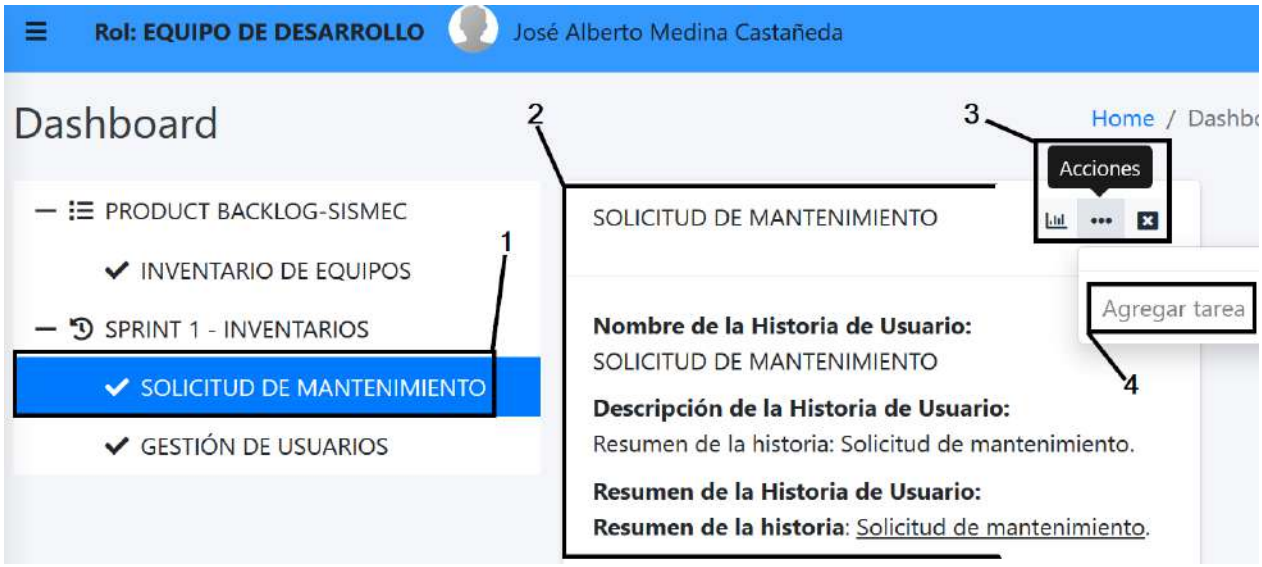


Figura 5.54. Vista mostrando el proceso para ver la opción Agregar tarea.

Al hacer clic en la opción **Agregar tarea** se mostrará un *modal pop-up* que contiene un formulario donde solicita un nombre, descripción, estimación de esfuerzo (horas) y tipo de trabajo que se realizará en la tarea (ver figura 5.55).

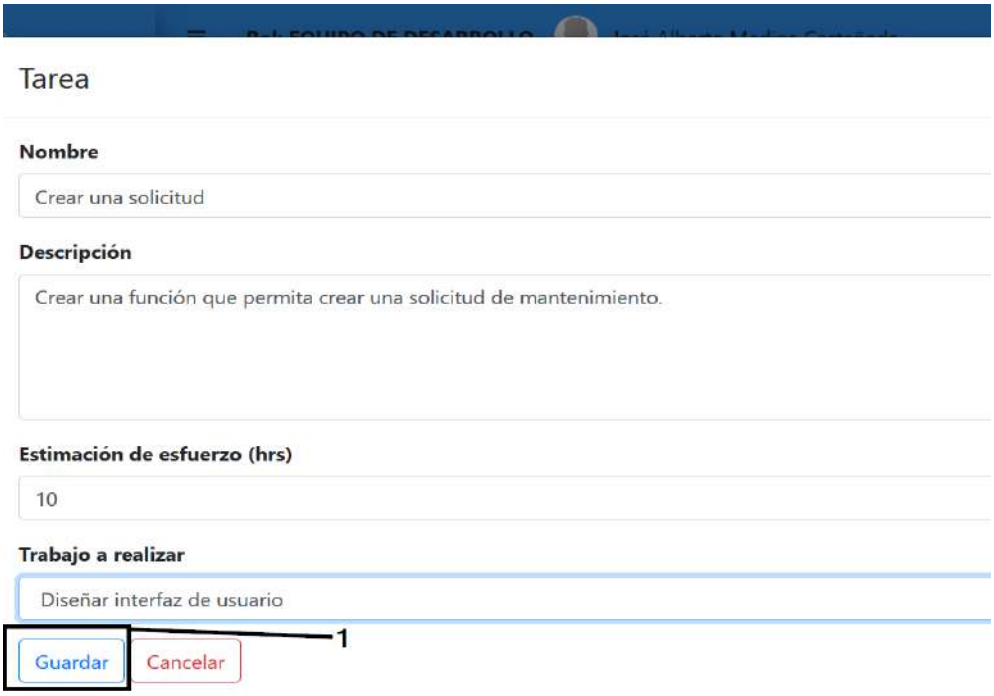


Figura 5.55. Formulario para agregar una tarea en una historia de usuario.

Al dar clic en el botón **Guardar** (ver indicador 1 en la figura 5.55) se enviarán los datos al servidor para almacenarlos en la base de datos y si todo se realiza correctamente, la aplicación mostrará un mensaje de éxito, como se muestra en la figura 5.56.

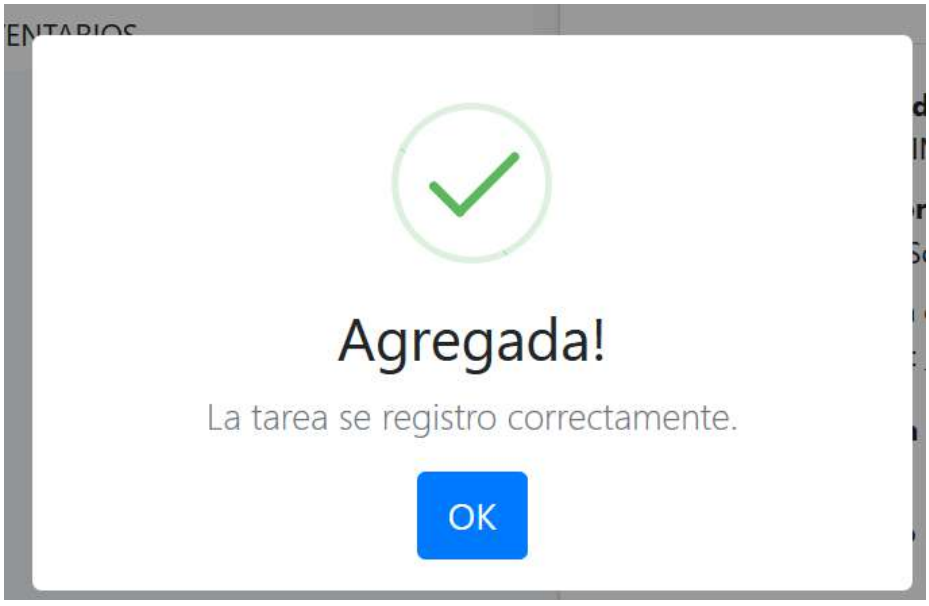


Figura 5.56. Mensaje de éxito cuando se agrega una tarea.

En la figura 5.57 se puede observar la tarea **Crear solicitud** que se creó (ver indicador 1), además de la su información en el recuadro ubicado en la parte derecha de la plataforma ***CyScrum*** (ver indicador 2).

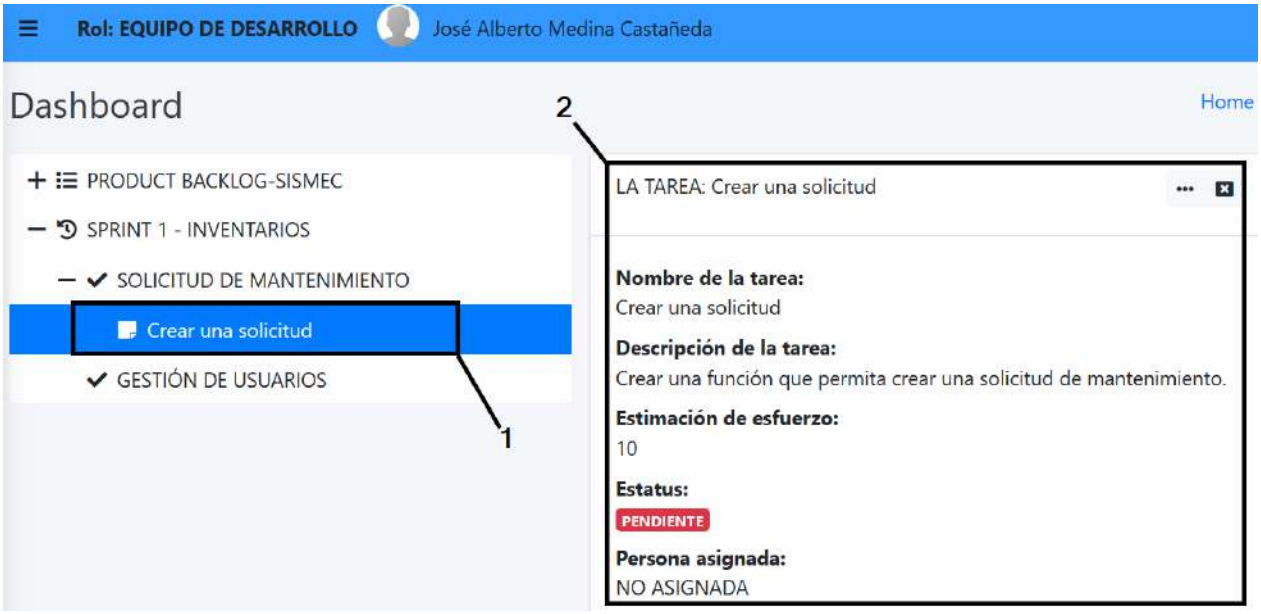


Figura 5.57. Verificación de la tarea Crear solicitud agregada.

6. Verificar en la base de datos el registro de la tarea creada.

Para verificar que se guardó correctamente la tarea creada en la base de datos se realizó una consulta a la tabla ***Tareas***, en la figura 5.58 se observa que la tarea **Crear solicitud** se guardó correctamente.

NombreTarea character varying(100)	RfcUsuario character varying(13)	TipoTRabajo character varying(70)	NombreHistoriaUsuario character varying(70)
Crear una solicitud		Diseñar interfaz de usuario	SOLICITUD DE MANTENIMIENTO

Figura 5.58. Consulta de los registros de las tareas en la tabla Tareas.

En la consulta realizada (ver figura 5.58) se puede observar que el campo **RfcUsuario** del registro de la tabla **Tareas** no cuentan con un valor. Esto es debido a que las tareas no tienen un a un integrante del equipo de desarrollo asignado. El proceso de asignación de tareas con base a la metodología Scrum, los mismos integrantes del equipo de desarrollo son quieren

eligen las tareas a realizar. En conclusión, el usuario con rol de equipo de desarrollo es quien tiene que asignarse las tareas que pretenda realizar.

7. Asignar la tarea creada al usuario **José Alberto** (**HU 15**).

Para que un usuario con rol de **equipo de desarrollo** se asigne una tarea, primero es necesario consultar la información de la **tarea**, la cual es mostrada en un recuadro (ver indicador 1) que además contiene las opciones **Editar**, **Eliminar** y **Asignar tarea** al hacer clic en el icono de los tres puntos (ver indicador 2), como se observa en la figura 5.59.

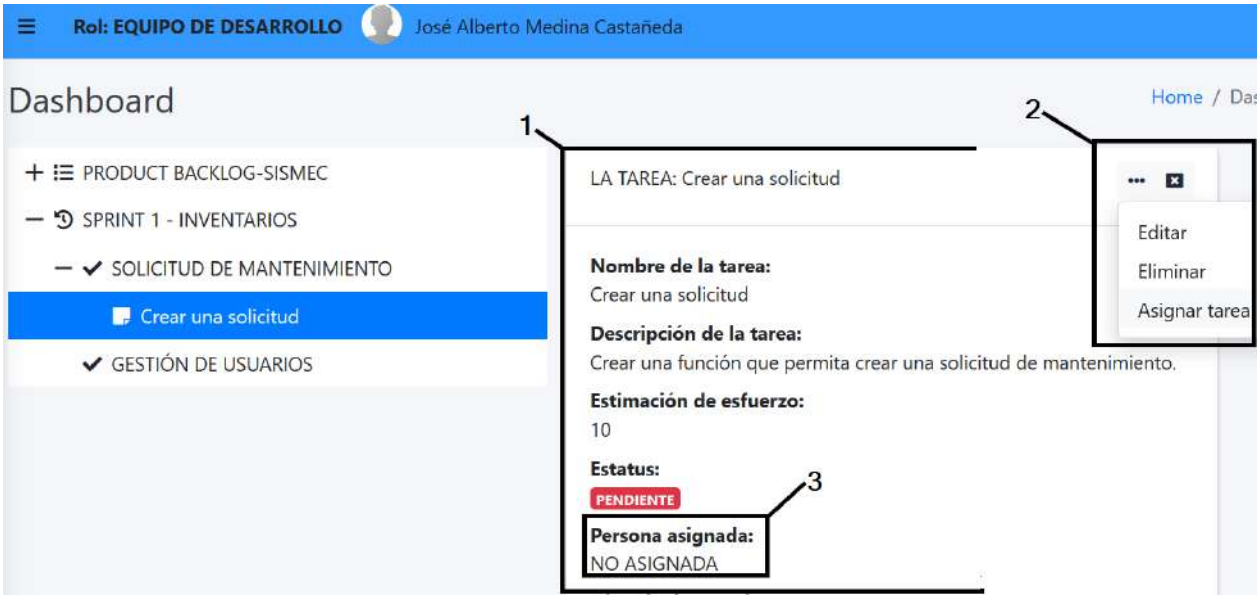


Figura 5.59. Área de trabajo para gestionar historias de usuario, Sprints y tareas en un proyecto.

Al momento de hacer clic en la opción **Asignar tarea** (ver indicador 2 en la figura 5.59), la plataforma ***CySCrum*** muestra un mensaje de confirmación de la acción a realizar en una alerta como se muestra en la figura 5.60.

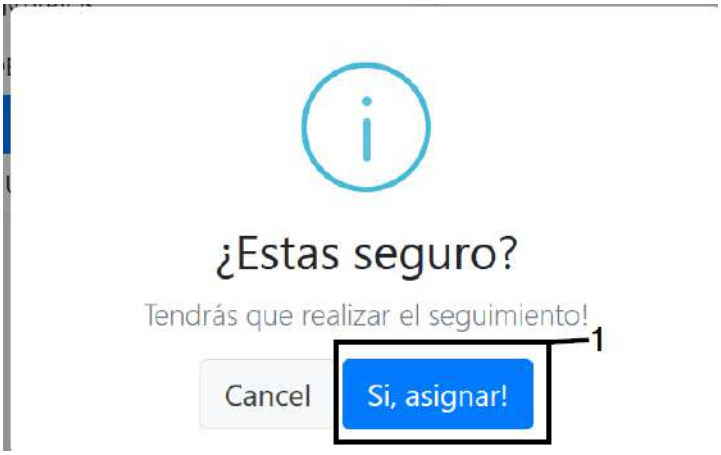


Figura 5.60. Mensaje de confirmación para asignar una tarea.

Cuando se selecciona el botón **si, asignar** (ver indicador 1 en la figura 5.60), la plataforma ***CyScrum*** visualiza un mensaje de éxito indicando que la tarea fue asignada, como se observa en la figura 5.61.

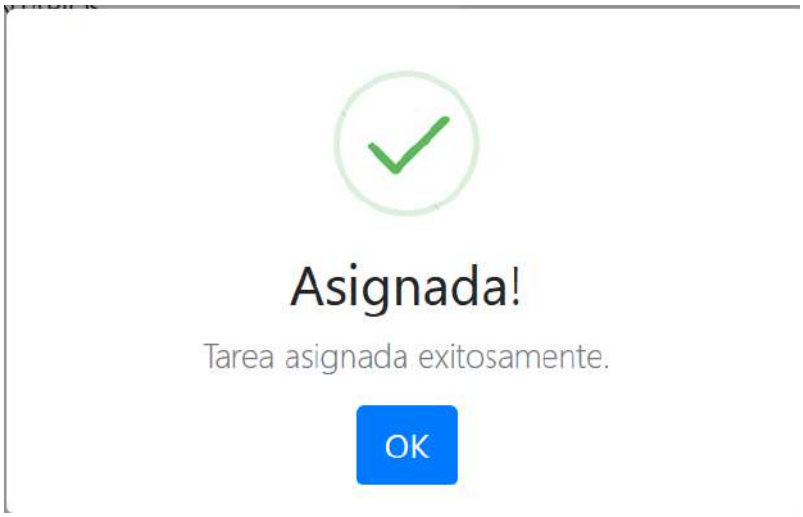


Figura 5.61. Mensaje de éxito cuando se asigna una tarea.

De esta forma se asignó la tarea **Crear solicitud** al usuario **José Alberto** con rol de equipo de desarrollo. En la figura 5.62 se puede observar en la información consultada de la tarea **Crear solicitud**, que en **Persona asignada** tiene el nombre del usuario **José Alberto** ver indicador (β).

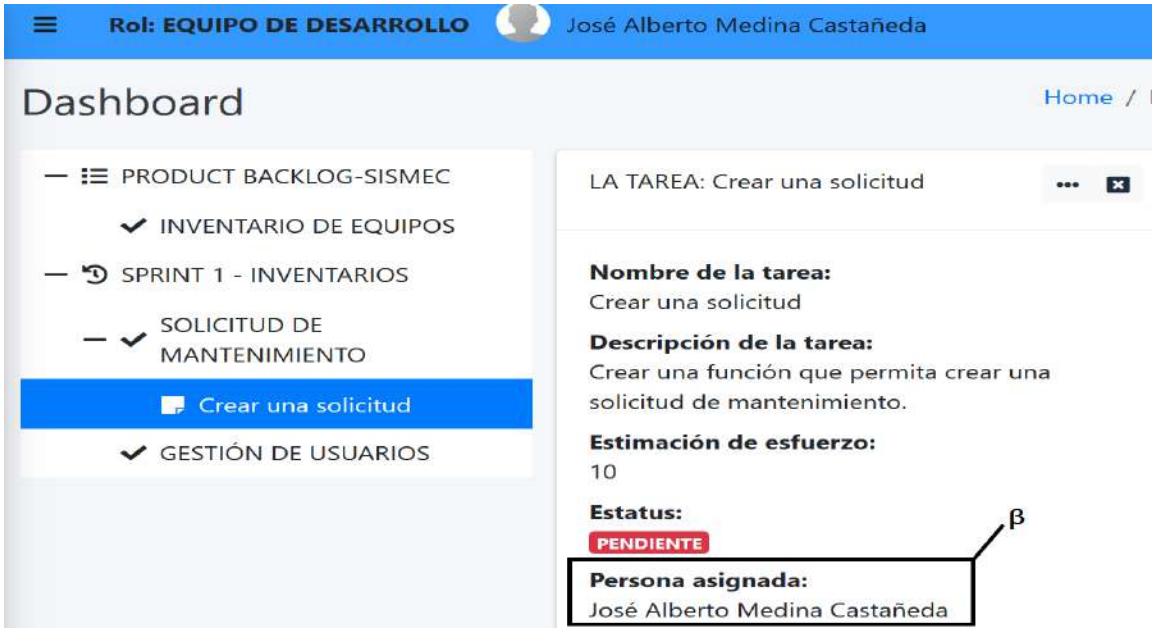


Figura 5.62. Consulta de información de una tarea.

8. Verificar en la base de datos que la información de la tarea se actualizó al momento de asignarle un integrante del equipo de desarrollo.

Al consultar en la base de datos el registro de la tarea **Crear solicitud** perteneciente a la historia de usuario **SOLICITUD DE MANTENIMIENTO**, se puede observar que tiene asignado el usuario **José Alberto** en el campo ***RfcUsuario*** (ver figura 5.63).

NombreTarea character varying(100)	Nombre character varying(70)	TipoTRabajo character varying(70)	NombreHistoriaUsuario character varying(70)
Crear una solicitud	José Alberto	Diseñar interfaz de usuario	SOLICITUD DE MANTENIMIENTO

Figura 5.63. Consulta del registro de la tarea para confirmar su asignación.

9. Crear un tipo de trabajo en el catálogo de tipos de trabajos del proyecto **SISMEC** (**HU 16**).

Para crear un tipo de trabajo a realizar en una tarea, lo primero es hacer clic en la opción **Tipos de trabajos** (ver indicador 1) que se encuentra en el panel lateral izquierdo de la vista mostrada en la figura 5.64. Este proceso es necesario debido a que cuando se agrega una tarea es necesario indicar que actividad o trabajo va a realizar el integrante del equipo de desarrollo asignado en la tarea.

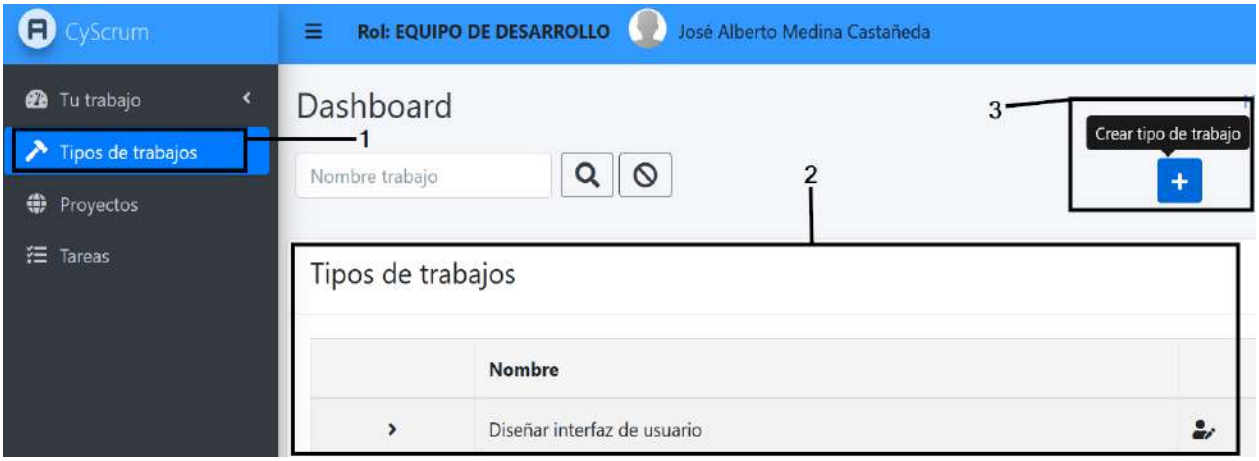


Figura 5.64. Vista para visualizar los tipos de trabajos.

Al hacer clic en la opción Tipos de trabajos, se muestra una vista con una tabla que contiene los tipos de trabajos registrados en un proyecto (ver indicador 2 en la figura 5.64). Para agregar un tipo de trabajo, se tiene que hacer clic en el botón con leyenda **Crear tipo de trabajo** (ver indicador 3), al momento de hacer clic en el botón mencionado, la plataforma ***CyScrum*** muestra un *modal pop-up* que contiene un formulario, donde simplemente solicita un nombre que tendrá el tipo de trabajo a crear (ver figura 5.65).

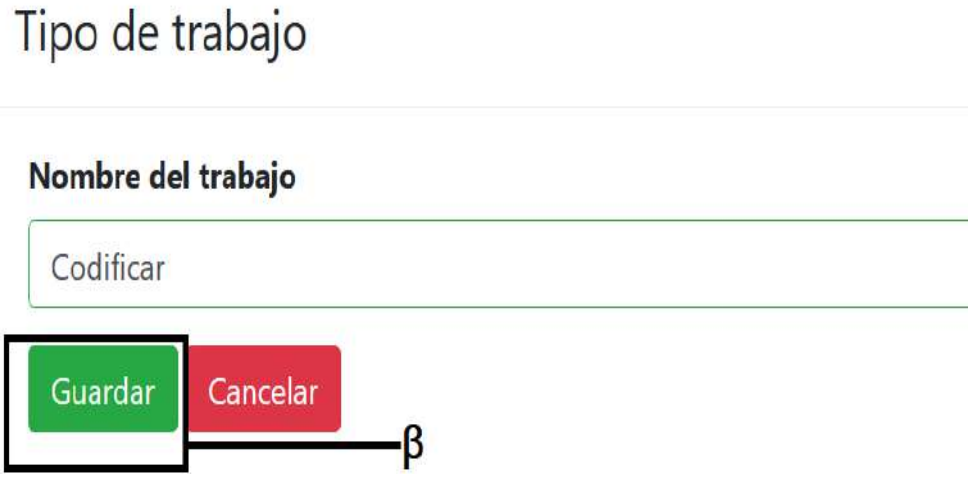


Figura 5.65. Formulario para crear un tipo de trabajo.

Después de haber introducido el nombre del trabajo, se hace clic en el botón **Guardar** (β en la figura 5.65) para enviar los datos al servidor y ser almacenados en la base de datos. Al

momento de hacer clic en el botón **Guardar**, la plataforma muestra un mensaje éxito cuando la acción se realiza correctamente (ver figura 5.66).

Agregado!

El tipo de trabajo se agrego exitosamente.

OK

Figura 5.66. Mensaje de éxito al hacer summit en el formulario para agregar un tipo de trabajo.

Cuando se realiza la transacción anterior, la tabla de tipos de trabajos se actualiza automáticamente, entonces se reflejará el nuevo tipo de trabajo agregado (β), como se muestra en la figura 5.67.

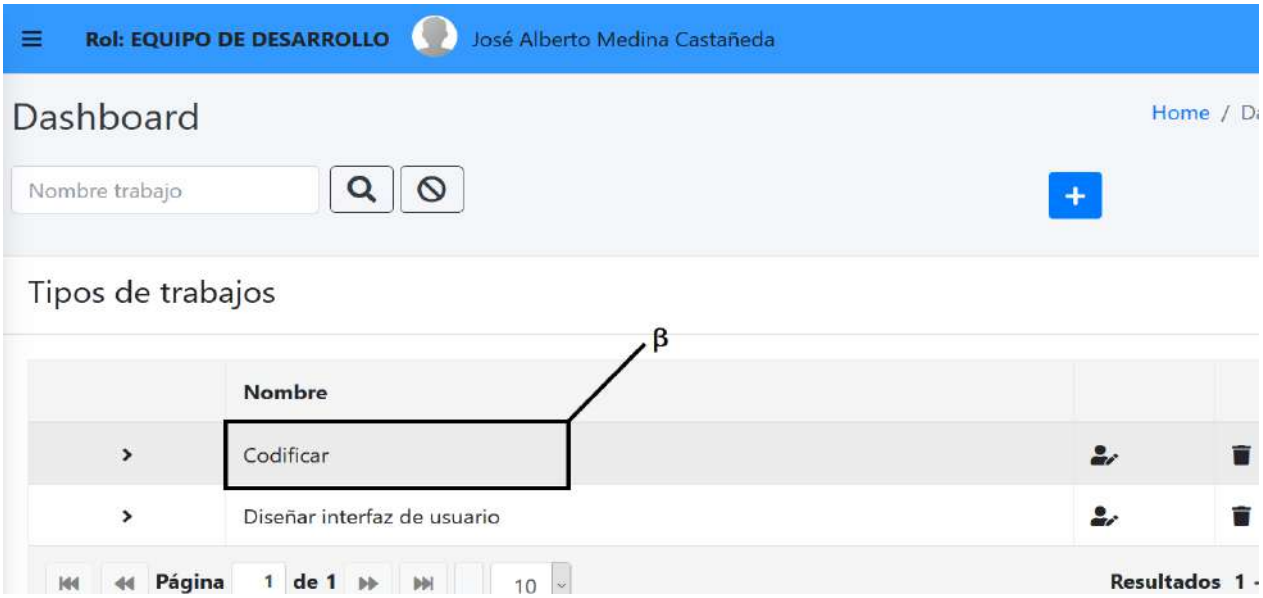


Figura 5.67. Vista de la lista de tipos de trabajos agregados.

10. Verificar en la base de datos que el tipo de trabajo creado se guardó correctamente.

Para verificar que el tipo de trabajo se registró correctamente en la base de datos, se realizó una consulta en la tabla tipos de trabajos. En la figura 5.68 se puede observar que el tipo de trabajo **Codificar** se guardó correctamente.

NombreCortoProyecto character varying(25)	NombreTrabajo character varying(70)
SISMEC	Diseñar interfaz de usuario
SISMEC	Codificar

Figura 5.68. Registros de los tipos de trabajo.

Conclusión

Se logró cumplir con los objetivos establecidos en este caso de estudio de manera satisfactoria, realizándose la creación de tareas en las historias de usuario contenidas en un sprint del proyecto Sistema de solicitudes. Así como también se verifico el correcto funcionamiento de la asignación de tareas al equipo de desarrollo. Además del cumplimiento de las HU 11 a HU 16 que fueron implementadas en el Sprint 4.

5.6 Caso de estudio 5: Seguimiento del Sprint

Descripción

En el presente caso de estudio se validará el funcionamiento para registrar el tiempo invertido en una tarea asignada a un integrante del equipo de desarrollo. Además, se validará el funcionamiento para generar el **burndow chart** a nivel Sprint.

Objetivos

1. Registrar el tiempo invertido en una tarea asignada por el usuario con rol de equipo de desarrollo.
2. Verificar en la base de datos si se guardó correctamente, el registro del tiempo invertido en una tarea.
3. Generar el **burndown chart** del seguimiento de un Sprint.

Procedimiento

1. Abrir el navegador web FireFox Mozilla.
2. Dirigirse a https://localhost:44367/.
3. Iniciar sesión con el usuario **José Alberto**.
4. Seleccionar el proyecto **SISMEC** que fue creado previamente.
5. Registrar el tiempo invertido en una tarea creada y asignada (**HU 17**).
6. Verificar en la base de datos que el registro del tiempo invertido en la tarea, se guardó correctamente.
7. Generar el burndown chart del seguimiento de un Sprint (**HU 18**).

Desarrollo

1. Abrir el navegador web FireFox Mozilla.

En la computadora con sistema operativo Windows 10 se procede abrir el navegador web FireFox Mozilla, dando clic en el icono de aplicación (ver figura 5.69).

Figura 5.69. Icono del navegador web FireFox Mozilla.

2. Dirigirse a https://localhost:44367/.

En la barra de navegación del navegador web FireFox se escribe la dirección https://localhost:44367 y se presiona la tecla [ENTER] (ver figura 5.70).

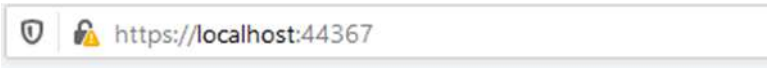


Figura 5.70. Barra de navegación de FireFox.

3. Iniciar sesión con el usuario **José Alberto**.

Figura 5.71. Vista para iniciar sesión en la plataforma CyScrum.

Para el registro del tiempo invertido en una tarea involucrada en un proyecto, es necesario acceder a un proyecto con rol de equipo de desarrollo, pero primero se necesita iniciar sesión. Como se observa en la figura 5.71, se introducen las credenciales del usuario **José Alberto** previamente creado y posteriormente se selecciona el botón **Iniciar sesión** (ver indicador 1).

4. Seleccionar el proyecto **SISMEC** que fue creado previamente.

Como se mencionó anteriormente es necesario acceder a un proyecto con rol de equipo de desarrollo establecido, para registrar el tiempo invertido en una tarea. Por tal motivo se elige el proyecto **SISMEC** (α), en el cual el usuario **José Alberto** fue invitado como colaborador con rol de equipo de desarrollo. Posteriormente de haber elegido el proyecto se selecciona el botón Seleccionar (β) (ver figura 5.72).

Figura 5.72. Vista para seleccionar un proyecto en la plataforma CyScrum.

Después de haber seleccionado el proyecto **SISMEC**, se muestra el product backlog con las historias de usuario, las tareas y el Sprint del proyecto **SISMEC** previamente creado, como se observa en la figura 5.73.

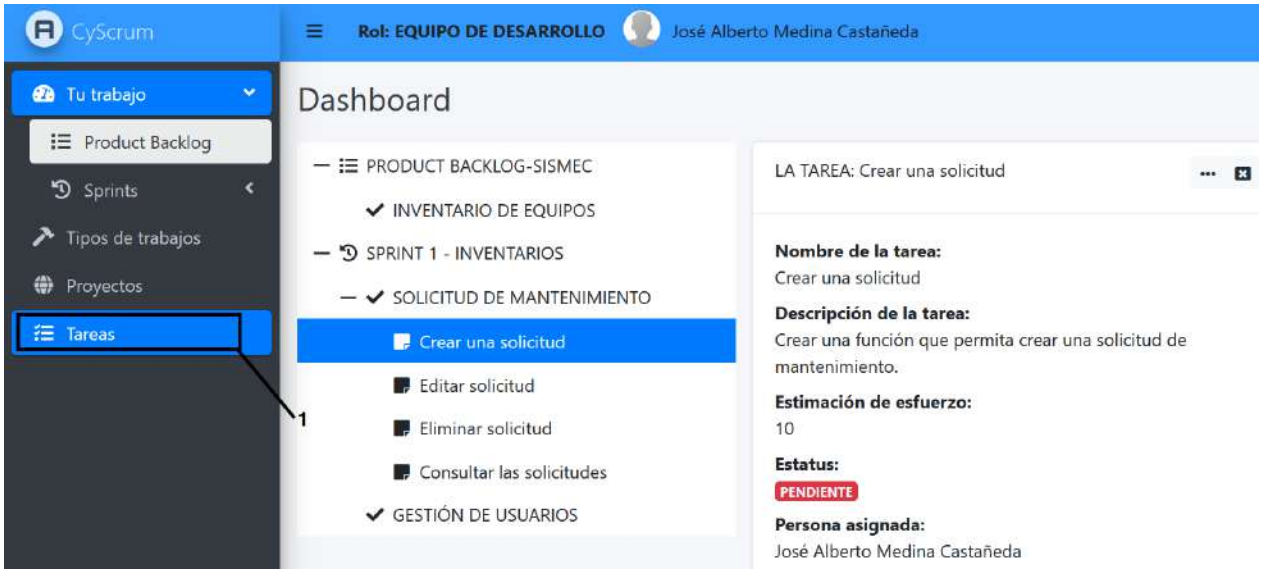


Figura 5.73. Vista principal de un proyecto a gestionar en la plataforma CyScrum.

5. Registrar el tiempo invertido en una tarea creada y asignada (**HU 17**).

Para poder registrar el tiempo invertido en las tareas asignadas a un integrante del equipo de desarrollo, se tiene que seleccionar la opción **Tareas** (ver indicador 1 en la figura 5.73).

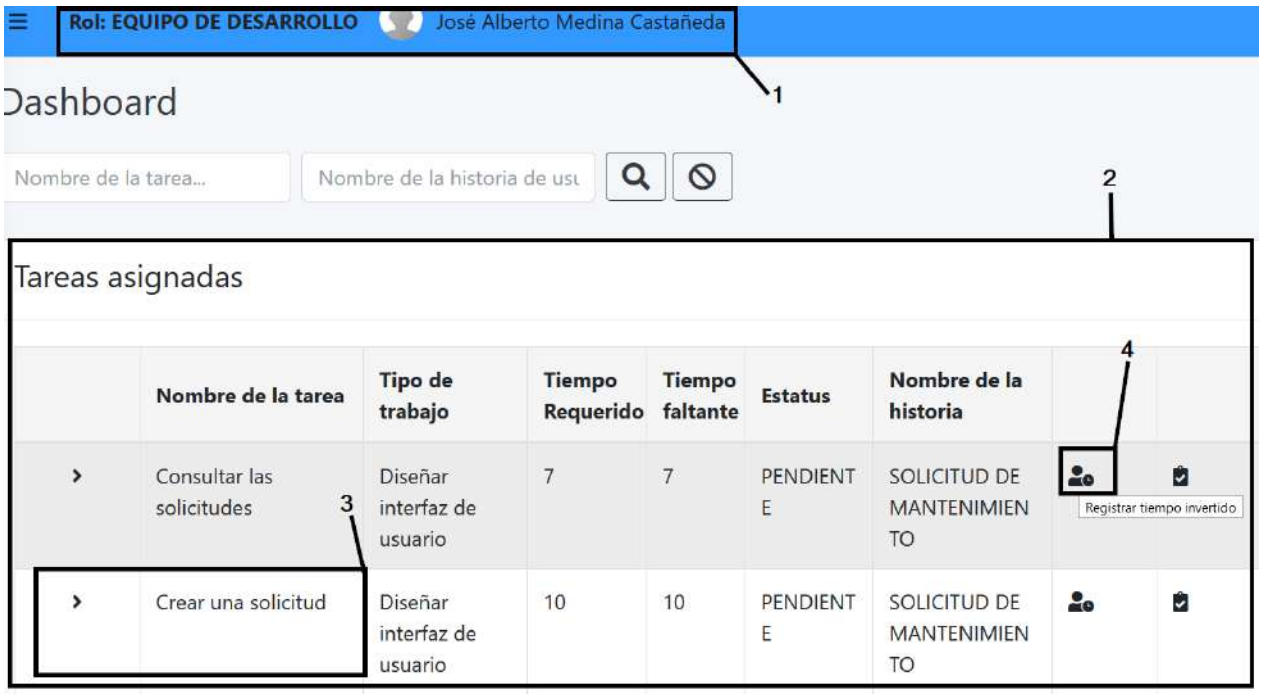


Figura 5.74. Vista para consultar las tareas asignadas.

Al seleccionar la opción **Tareas** se muestra una vista con una tabla (ver indicador 2 en la figura 5.74) que contiene las tareas asignadas al usuario con rol de equipo de desarrollo, en

este caso el usuario **José Alberto** (ver indicador 1). Cuando en una tarea aún no existe un registro del tiempo invertido su estatus esta en pendiente como la tarea **Crear una solicitud** (ver indicador 3) y para realizar esta tarea requiere 10 horas de inversión.

Para realizar un registro de tiempo invertido en una tarea, se hace clic en el icono que tiene la leyenda **Registrar tiempo invertido** (ver indicador 4). Al momento de realizar esta acción se muestra un *modal pop-up* que contiene un formulario para introducir el tiempo invertido y la fecha, como se muestra en la figura 5.75.

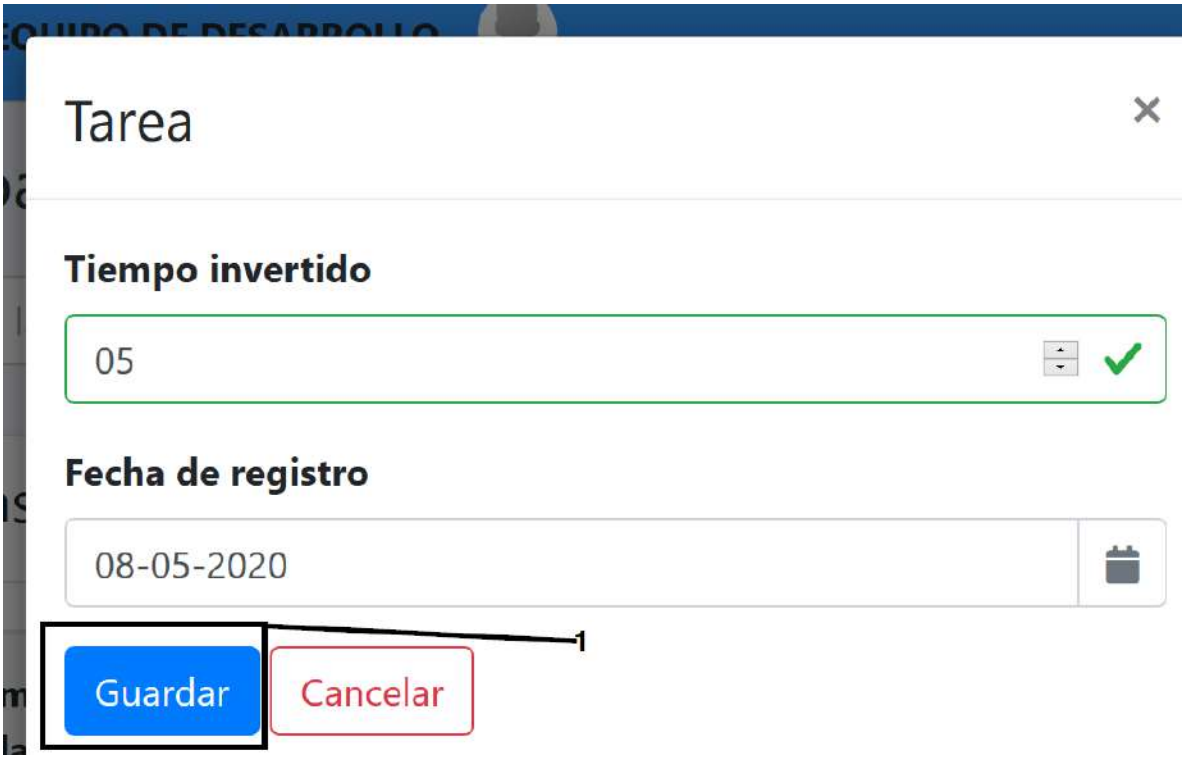


Figura 5.75. Formulario para registrar el tiempo invertido en una tarea.

El tiempo invertido registrado en el formulario que se muestra en la figura 5.75, es para la tarea **Crear una solicitud**. Después de haber introducido los datos necesarios, se hace clic en el botón **Guardar** (ver indicador 1). Al momento de hacer clic en el botón **Guardar** la plataforma envía un mensaje de éxito en caso de ser así (ver figura 5.76).

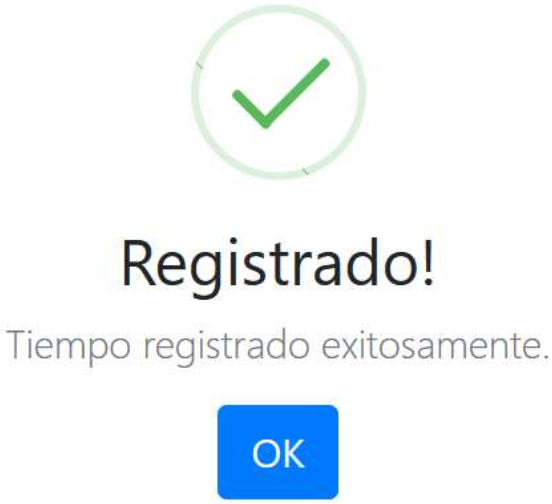


Figura 5.76. Mensaje de éxito al momento de registrar tiempo invertido en una tarea.

Después de realizar el registro del tiempo invertido en la tarea **Crear una solicitud** la tabla de las tareas asignadas se actualiza automáticamente, en la figura 5.77 se observa que la tarea **Crear una solicitud** (ver indicador 1), ahora en la columna **Tiempo faltante** tiene el valor de 5 indicando que son las horas restantes para finalizar la tarea, al realizar el registro del tiempo invertido en la tarea **Crear una solicitud** su estatus cambio a **PROCESO**.

Tareas asignadas

	Nombre de la tarea	Tipo de trabajo	Tiempo Requerido	Tiempo faltante	Estatus	Nombre de la historia		
›	Consultar las solicitudes	Diseñar interfaz de usuario	7	7	PENDIENTE	SOLICITUD DE MANTENIMIENTO		
1 ›	Crear una solicitud	Diseñar interfaz de usuario	10	5	PROCESO	SOLICITUD DE MANTENIMIENTO		

Figura 5.77. tabla de las tareas asignadas, actualización de información de una tarea.

Para demostrar que sucede cuando finaliza una tarea, se realizó otro registro de tiempo invertido en la tarea **Crear una solicitud**, donde se registró 5 horas. Después de realizar lo mencionado, en la figura 5.78 se observa como la tarea **Crear una solicitud** cambia su estatus a **FINALIZADO** (ver indicador 1). Además, cuando una tarea ha sido finalizada la opción de **Registrar tiempo invertido** no aparece.

Figura 5.78. Lista de tareas asignada, donde una ha sido finalizada.

6. Verificar en la base de datos que el registro del tiempo invertido en la tarea, se guardó correctamente.

NombreTarea character varying(100)	EstimacionEsfuerzo integer	EstimacionEsfuerzoRestante integer	Estatus character varying(70)	Nombre character varying(70)	TipoTRabajo character var
Crear una solicitud	10	0	FINALIZADO	José Alberto	Diseñar in
Consultar las solicitud	7	7	PENDIENTE	José Alberto	Diseñar in
Eliminar solicitud	15	15	PENDIENTE	José Alberto	Diseñar in
Editar solicitud	15	15	PENDIENTE	José Alberto	Diseñar in

Figura 5.79. Consulta delos registros de las tareas del proyecto SISMEC en la base de datos.

Para verificar que las transacciones (registros del tiempo invertido en la tarea **Crear una solicitud**) se realizaron correctamente, se consultaron los registros de las tablas ***Tareas*** (ver figura 5.79) y ***SeguimientoSprint*** (ver figura 5.80). En la figura 5.79 se observa que la actualización del estatus (FINALIZADO) de la tarea **Crear una solicitud** se realizó correctamente (ver indicador 1 en la figura 5.79).

En la figura 5.80 se muestran los registros del tiempo invertido por parte del usuario **José Alberto** en la tarea **Crear una solicitud**.

NombreTarea character varying(100)	FechaRegistro timestamp wit	TiempoInvertido integer	NombreSprint character varying(150)	Nombre character varyi
Crear una solicitud	2020-05-08 0	5	SPRINT 1 - INVENTARIOS	José Alberto
Crear una solicitud	2020-05-09 0	5	SPRINT 1 - INVENTARIOS	José Alberto

Figura 5.80. Registros del seguimiento del Sprint SPRINT 1 - INVENTARIOS.

7. Generar el burndown chart del seguimiento de un Sprint (**HU 18**).

Para consultar el burndown chart de un Sprint, es necesario seleccionar la opción **Tu trabajo** (ver indicador 1 en la figura 5.81) que se encuentra en el panel izquierdo, entonces se desplegará menú con dos opciones más (**Product backlog** y **Sprints**) (ver indicador 2), posteriormente seleccionar la opción **Sprints**, al hacer esto se mostrará la lista de Sprints (ver indicador 3) del proyecto gestionado y simplemente es seleccionar el Sprint deseado para consultar su burndown chart.

Figura 5.81. Procedimiento para consultar el burndown chart de un Sprint.

Para el ejemplo se seleccionó el Sprint con nombre **SPRINT 1 – INVENTARIOS** (ver indicador 3). Cuando se selecciona el Sprint **SPRINT 1 – INVENTARIOS**, la plataforma se re direcciona a la vista que se muestra en la figura 5.82.

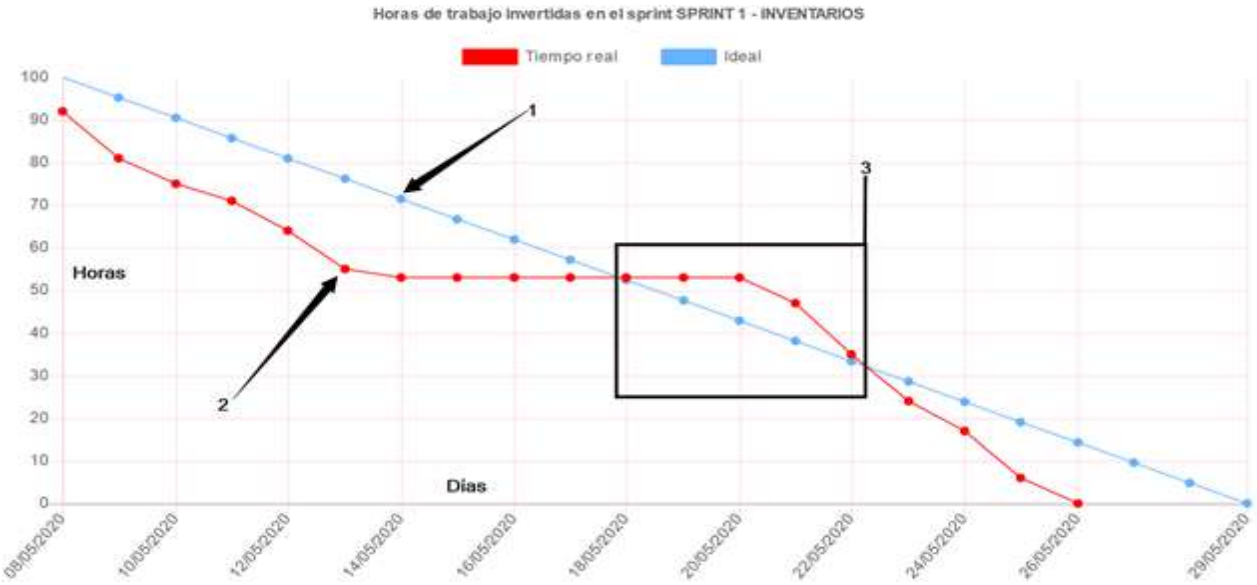


Figura 5.82. Vista para visualizar el burndown chart de un Sprint.

En la figura 5.82 se muestra el burndown chart del sprint con nombre **SPRINT 1-INVENTARIOS** donde se grafican dos líneas donde una es lo ideal (ver indicador 1), es decir, la representación del tiempo de inversión requerido para alcanzar la meta, o la posición ideal para estar al final de cada día. Lo real (ver indicador 2) es el tiempo invertido en el sprint por parta del equipo de desarrollo. Con este grafico se puede hacer una comparación de lo ideal contra lo real proporcionando una medida simple del progreso en el sprint.

En la figura 5.82 se puede observar que en un determinado momento lo real estuvo por encima de lo ideal (ver indicador 3), indicando que su momento hubo un retraso.

Conclusión

Se logró cumplir con los objetivos propuestos en este caso de estudio de manera satisfactoria, realizando el registro del tiempo invertido en una tarea asignada a un usuario con rol de equipo de desarrollo, así como la verificación de estos registros en la base de datos, donde se realizaron correctamente. Además, realizó la consulta del burndown chart de un sprint permitiendo ver el progreso lo cual ayuda a saber si se cumplirán los objetivos establecidos en el sprint en tiempo y forma. Además del cumplimiento de las **HU 17** y **HU 18** que fueron desarrolladas en el quinto Sprint presentado en el capítulo 4.

Capítulo 6 Conclusiones

El desarrollo de la plataforma ***CyScrum*** permite solventar la ausencia de una herramienta para el monitoreo y gestión de proyectos de desarrollo de software basado en la metodología Scrum, en el departamento de desarrollo de software de la empresa **PICCE**. Como se describió en el capítulo 5, donde los casos de estudio fueron cumplidos a cabalidad las historias de usuario especificadas en el capítulo 3 de la HU 1 a la HU 18, solventado así la parte metodológica Scrum en cuanto al registro de historias de usuario y tareas, el registro del tiempo invertido en cada una de las tareas involucradas en el desarrollo de software, así como la generación del **burndown chart** y la asignación de responsabilidades con base al rol.

A través del caso de estudio 5 presentado en el capítulo 5 es posible comprobar la hipótesis planteada, donde se visualiza la gestión y monitorización de las tareas del proyecto ***SISMEC*** con la plataforma ***CyScrum***, dando lugar a la generación del **burndown chart** donde se puede observar gráficamente el avance real contra lo ideal, permitiendo hacer estimaciones para alcanzar los objetivos en cada Sprint y a su vez obtener entregas a tiempo.

Con el uso de las herramientas de desarrollo descritas en el capítulo 2 fue posible desarrollar una plataforma web que permite trabajar con múltiples usuarios a la vez, sin afectar el tiempo de respuesta debido a la implementación de los patrones unidad de trabajo y repositorio, así como la implementación del patrón arquitectónico MVC, dando la posibilidad de escalar la plataforma en caso de ser necesario. La plataforma ***CyScrum*** no necesita de una configuración experta como otras plataformas existentes de gestión de proyecto, debido a que es una plataforma web y para el acceso únicamente es necesario de un navegador web.

Cabe mencionar que no es recomendable utilizar la plataforma para la gestión de proyectos que no sean de desarrollo de software basado en la metodología Scrum, debido a que la plataforma ***CyScrum*** sigue estrictamente la metodología Scrum, esto para garantizar las ventajas proporcionas por la metodología Scrum.

6.1 Trabajos a futuro

En los trabajos a futuro contemplados para las posibilidades de acceso y uso de la plataforma ***CyScrum***, es la migración del **Back-End** de ***CyScrum*** a una **API Rest**, permitiendo la creación servicios **RESTful HTTP**. De esta forma proporciona las siguientes ventajas:

- Solamente se enfocaría a la implementación del **Front-End** para diferentes tipos de aplicaciones (web, móvil o escritorio).
- Da la posibilidad de crear el **Front-End** con casi cualquier lenguaje, aprovechando las bondades de este para la creación de algún tipo de aplicación utilizando los servicios proporcionados por una **API Rest**.
- Expande las posibilidades de implementar diferentes tipos de seguridad el acceso los servicios proporcionados por la API.

Como trabajo a futuro también se pueden considerar las siguientes funcionalidades:

1. Permita al **product owner** subir archivos de Word o texto plano en las historias de usuario o tareas, esto para proporcionar información más detallada.
2. Que tenga un diagrama de Gantt con el objetivo de exponer el tiempo de dedicación previsto para las diferentes actividades que estarán involucradas en un proyecto.

3. Integrar la API de Git para que de esta forma se garante la finalización de las tareas involucradas en el desarrollo de software a través del control de versiones y también permitiendo la integración continua.
4. Que tenga un sistema de notificaciones, donde se pueda configurar a quien y para que acción se realizarán notificaciones.
5. Permita el manejo de grupos entre los integrantes del equipo de desarrollo, permitiendo la asignación de Sprint a estos grupos, lo cual dará la posibilidad de evaluar y comparar el desempeño de los grupos.

Participaciones y aportaciones

Durante la estancia en la Maestría en Sistemas Computacionales impartida en el Tecnológico Nacional de México campus Acapulco con apoyo de CONACyT, se realizaron las siguientes participaciones y aportaciones en diferentes ámbitos derivadas de este proyecto:

1. Publicación de cartel en Expo-Proyecta (Tecnológico Nacional de México, campus Acapulco).
2. Artículo y ponencia en el Congreso Internacional de Investigación de Academia Journals Morelia 2019.
3. Artículo y ponencia en el Congreso Internacional de Investigación de Academia Journals Puebla 2019.
4. Artículo y ponencia en el Séptimo Congreso Internacional de Robótica y Computación del Tecnológico Nacional de México campus La paz 2020.
5. Artículo publicado en la Séptima edición, No. 2 Vol. 1 de la revista Ingeniantes noviembre 2020, LATINDEX 25671.
6. Estancia realizada en PICCE, donde fueron entregados:
    a. Aplicación Web (plataforma CyScrum).
    b. Manual de usuario.
    c. Manual técnico.

Referencias

Aksit, M., Tekinerdogan, B., & Bergmans, L. (2001). The Six Concerns for Separation of Concerns. *ECOOP*.

Beck, K. (2000). *Extreme Programming Explained. Embrace Change.* Addison Wesley.

Benedicenti, L., Cotugno, F., Ciancarini, P., Angelo, M., Pedrycz, W., Sillitti, A., & Succi, G. (2016). Applying Scrum to the Army - A Case Study. *IEEE Xplore*, 725-727.

Bipin, J. (2016). *Beginning SOLID principles and design patterns for ASP.NET developers.* Thane, India: Apress.

Blankenship, J., Bussa, M., & Millett, S. (2011). *Pro Agile .Net Development with Scrum.* Apress.

Bosch, J. (2000). *Design & Use of Software Architectures.* Addison-Wesley.

Ceballos Sierra, F. J. (2013). *Enciclopedia de Microsoft Visual C#: Interfaces gráficas y aplicaciones para internet con Windows Form y ASP.NET.* Madrid: RA-MA.

Ceballos, F. J. (2007). *Enciclopedia de Microsoft: Visual C#.* México: Alfaomega.

Ceballos, F. J. (2012). *Microsoft C#: Curso de programación.* Mexico D.F: Ra-Ma.

Cervantes Maceda, H., Velasco-Elizondo, P., & Castro Careaga, L. (2016). *Arquitectura de software: Conceptos y ciclo de desarrollo.* Ciudad de México: Cengage Learning.

Coad, P., Lefebvre, E., & De Luca, J. (1999). *Java Modeling In Color With UML: Enterprise Components and Process.* Prentice Hall.

Cockburn, A. (2001). *Agile Software Development.* Addison-Wesley.

Cohen, M. (2004). *User Stories Applied for Agile Software Develoment.* The Addison-Wesley Signature Series.

Freeman, A. (2016). *Pro ASP.NET Core MVC: Develop cloud-ready web applications using Microsoft's latest framework, ASP.NET Core MVC.* New York: Apress.

Freeman, A. (2018). *Pro Entity Framework Core 2 for ASP.NET Core Mvc.* Londo, UK: Apress.

Freeman, A., & Rattz, J. C. (2010). *LINQ Languaje Integrated Query in C# 2010.* United States of America: Apress.

Godoy, D. A. (2014). Diseño de un simulador dinámico de proyectos de desarrollo de software que utilizan metología Scrum. España: Universidad Nacional de La Plata Facultad de Informática.

González, P. R. (Septiembre de 2008). Estudio de la aplicación de metodologías ágiles para la evolución de productos software. Madrid, España.

Herrera, U., Valencia, A., & Luz, E. (2007). Del manifiesto ágil sus valores y principios. *Scientia Et Technica*, 381-386.

Highsmith, J. (2002). *Agile Software Development Ecosystems.* Addison Wesley.

James A., H., & Orr, K. (2000). *Adaptive Software Development: A Collaborative Approach to Managing Complex Systems.* Dorset House.

Letelier, P., & Panadés, C. (2012). Metodologías ágiles para el desarrollo de software: eXtreme Programming (XP).

Matthew, N., & Stones, R. (2005). *Beginning Databases with PostgreSQL: From novice to professional.* United States of America: Apress.

Munro, J. (2015). *ASP.NET MVC 5 with Bootstrap and Knockout.js.* United States of America: O'Reilly.

Nixon, R. (2012). *Learning PHP, MySQL, JavaScript y CSS.* United States of America: O'Reilly.

Poppendieck, M., & Poppendieck, T. (2003). *Lean Software Development: An Agile Toolkit for Software Development Managers.* Addison Wesley.

Refsnes Data. (2020). Obtenido de w3school.com.

Resig, J. (26 de Agosto de 2006). *Jquery*. Obtenido de Jquery: https://jquery.com

Rumbaugh, J., Jacobson, I., & Booch, G. (2000). *El lenguaje unificado de modelado. Manual de referencia.* Madrid: Addison Wesley.

Sommerville, I. (2016). *Ingenieria de Software.* México: Pearson educacion.

Stapleton, J. (1997). *Dsdm Dynamic Systems Development Method: The Method in Practice.* Addison-Wesley.

Tahuiton Mora, J. (2011). *Arquitectura de software para aplicaciones Web.* Unidad Zacatento .

Twitter. (19 de Agosto de 2011). *Bootstrap*. Obtenido de Bootstrap: https://www.getbootstrap.com/