

Universidad ORT Uruguay
Facultad de Ingeniería

ChefCheck-Manager

Sistema administrativo de alimentos y bebidas para hotel *All inclusive Tío Tom*

Entregado como requisito para la obtención del título de Licenciatura en Sistemas

Francisco Cabanillas – 231918

Andrés Quintero - 233813

Martín Sosa – 228368

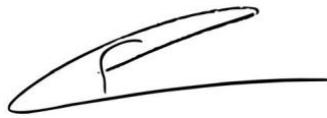
Tutor: Helena Garbarino

2024

Declaración de autoría

Nosotros, Francisco Cabanillas, Andrés Quintero y Martín Sosa, declaramos que el trabajo que se presenta en esa obra es de nuestra propia mano. Podemos asegurar que:

- La obra fue producida en su totalidad mientras realizábamos el proyecto final de la carrera Licenciatura en Sistemas;
- Cuando hemos consultado el trabajo publicado por otros, lo hemos atribuido con claridad;
- Cuando hemos citado obras de otros, hemos indicado las fuentes. Con excepción de estas citas, la obra es enteramente nuestra;
- En la obra, hemos acusado recibo de las ayudas recibidas;
- Cuando la obra se basa en trabajo realizado conjuntamente con otros, hemos explicado claramente qué fue contribuido por otros, y qué fue contribuido por nosotros;
- Ninguna parte de este trabajo ha sido publicada previamente a su entrega, excepto donde se han realizado las aclaraciones correspondientes.



Francisco Cabanillas

21 de Octubre de 2024



Andrés Quintero

21 de Octubre de 2024



Martín Sosa

21 de Octubre de 2024

Agradecimientos

Queremos agradecer de manera especial a Helena Garbarino, quien se desempeñó como nuestra tutora, por su gran apoyo y orientación durante todos los meses del desarrollo de este proyecto. Su amplia experiencia y capacidad para guiarnos fueron esenciales para lograr nuestros objetivos.

También queremos agradecer a la Universidad ORT Uruguay, que nos ha proporcionado las herramientas y el entorno necesarios para nuestra formación a lo largo de los cuatro años de carrera. A todos los docentes que nos acompañaron a lo largo del trayecto, así como al personal administrativo, les estamos profundamente agradecidos por su contribución y compromiso.

A Gastón Mousqués, le extendemos nuestro más sincero agradecimiento por su asesoramiento técnico en lo que corresponde a la arquitectura del sistema. Sus consejos fueron fundamentales para la estructura y el éxito del proyecto.

A Diego Dodel, de FacturaLista quien nos brindó su apoyo para resolver todas nuestras dudas e inconvenientes en cuanto a la integración con el sistema de dicha empresa.

También le queremos agradecer a Marcelo Cagnani, quien se desempeñó como revisor del equipo, otorgando *feedback* de gran valor.

Un agradecimiento especial a Maximiliano Correa, gerente del hotel Tío Tom *All Inclusive*, por su cálida recepción y generosidad al permitirnos conocer en primera persona la operativa del hotel. Su disposición y confianza en nosotros fueron la clave para entender y adaptar nuestra solución a las necesidades reales del negocio.

Finalmente, a nuestras familias y amigos, queremos expresarles nuestro más profundo agradecimiento por el apoyo incondicional que nos brindaron. Su paciencia, aliento y compañía nos motivaron a seguir adelante y completar este trabajo.

Abstract

El proyecto denominado *ChefCheck-Manager*, es una solución que tiene como objetivo abordar los desafíos de la gestión de inventario y el control de costos para el cliente hotel Tío Tom *All Inclusive*. Ofrece una solución para automatizar los procesos de gestión de alimentos y bebidas.

En un entorno donde la precisión de los datos y la eficiencia de los procesos son fundamentales para la rentabilidad y el éxito de la modalidad *all inclusive*, este sistema proporciona una herramienta que puede ser utilizada tanto en dispositivos móviles como en navegadores *web*, asegurando que la gestión de los recursos sea lo más preciso posible.

Previo al desarrollo del sistema, el equipo comenzó con una investigación detallada de las necesidades y requerimientos específicos del hotel, lo cual permitió definir un producto mínimo viable, el cual aborda aquellos requerimientos de mayor prioridad. Entre las funcionalidades clave se incluyen el control en tiempo real del inventario, el registro detallado de los movimientos de stock/inventario y la integración automatizada con la empresa FacturaLista, para gestionar las facturas electrónicas de manera eficiente. Este enfoque no solo mejora la precisión en la gestión de costos, sino que también les facilita a los altos cargos, la toma de decisiones basada en datos en tiempo real, reduciendo la intervención manual y minimizando errores y tiempo perdido.

Respecto a la arquitectura, el equipo afirma que el sistema se construye bajo un enfoque de microservicios que utiliza tecnologías como .NET 8.0 para el *backend*, Angular versión 18.0 para el *frontend web* y Flutter 3.19.16 para la aplicación de tablets y celulares. Esta arquitectura permite que cada componente del sistema funcione de manera independiente, lo que facilita el mantenimiento y la incorporación de nuevas funcionalidades. La flexibilidad del sistema permite que se pueda adaptar a futuras expansiones, de acuerdo con las necesidades que puedan surgir en la operación diaria del hotel. Además, gracias al diseño modular y escalable del sistema, se asegura que la solución pueda integrarse fácilmente con otros sistemas, brindando una mayor capacidad de respuesta ante las demandas existentes.

Durante el desarrollo, se priorizaron aspectos que el equipo consideró como relevantes, como, por ejemplo, la usabilidad y la accesibilidad, asegurando que el sistema sea intuitivo y fácil de utilizar por parte del personal del hotel. Las funcionalidades de automatización reducen significativamente la carga de trabajo manual, permitiendo que el equipo del hotel se pueda concentrar en otras tareas. A medida que el hotel crezca o evolucionen sus necesidades, el sistema está preparado para adaptarse sin la necesidad de realizar cambios profundos en su estructura, garantizando su vigencia a largo plazo.

Palabras clave

MVP, Sistema de Gestión, Inventario, Análisis de Costos, Hotel, All Inclusive, Microservicios, PostgreSQL, Angular, Flutter, .NET, Machine Learning, Factura Electrónica, FacturaLista, Azure, API REST, Seguridad, Escalabilidad, Scrum, Ciclo de Vida Incremental, DevOps, Gestión de Calidad, JIRA, Automatización, Firebase, SP (Story Point).

1.	Introducción	11
1.1.	Objetivos del proyecto	11
1.1.1.	Proyecto	11
1.1.2.	Producto	12
1.1.3.	Académicos.....	12
1.1.4.	Equipo	13
2.	Problema y su solución	14
2.1.	Introducción	14
2.2.	Problema	14
2.3.	Solución	15
3.	Marco metodológico	17
3.1.	Ciclo de vida	17
3.2.	Metodologías ágiles	18
3.2.1.	Scrum	18
3.2.2.	Comparativa Scrum vs Kanban	18
3.2.3.	Descartar Kanban	19
3.3.	Etapas del proyecto	19
3.3.1.	Investigación y análisis	19
3.3.2.	Desarrollo y pruebas	19
3.3.3.	Fiscalización y mejora continua del producto	20
3.4.	Roles	20
3.5.	Cronograma del proyecto.....	21
3.5.1.	Verificación de requerimientos con el cliente	22
3.5.2.	Entrega del anteproyecto.....	23
3.5.3.	Revisión del proyecto	23
3.5.4.	Revisión de la arquitectura.....	23
3.5.5.	Entrega administrativa en el portal de gestión	23
3.5.6.	Defensa	23
3.5.7.	Puesta en producción	23
3.5.8.	Fin de soporte de garantía	24
4.	Ingeniería de requerimientos.....	25

4.1.	Obtención y análisis de requerimientos	25
4.1.1.	Herramientas para el relevamiento	25
4.1.2.	Conclusión	28
4.2.	Especificación de requerimientos	29
4.2.1.	Prototipo "Finanzas Tío Tom"	29
4.3.	Lista de requerimientos.....	32
4.3.1.	Historias de usuario.....	32
4.3.2.	Requerimientos principales funcionales	33
4.3.3.	Requerimientos no funcionales	36
4.3.4.	Entregables.....	38
4.4.	Validación de problema y solución.....	39
5.	Arquitectura de la solución	40
5.1.	Introducción a la arquitectura	40
5.2.	Arquitectura basada en microservicios	41
5.2.1.	Justificación de la utilización de una arquitectura basada en microservicios	41
5.2.2.	Arquitectura a alto nivel.....	43
5.2.3.	Importancia de la separación en tres componentes (<i>frontend</i> , <i>backend</i> y servicio de FacturaLista)	45
5.2.4.	Descripción de la comunicación entre componentes	47
5.3.	Plataformas tecnológicas y justificaciones de diseño	52
5.3.1.	Elección de tecnologías y justificación.....	52
5.4.	Arquitectura interna de cada componente.....	57
5.4.1.	<i>Frontend</i>	57
5.4.2.	<i>Backend</i>	62
5.5.	Seguridad del sistema	81
5.5.1.	Seguridad en el Transporte de la Información	81
5.6.	<i>Deployment</i> en Azure	81
5.6.1.	Base de datos PostgreSQL	82
5.6.2.	<i>API .NET</i>	83
5.6.3.	<i>Web App Angular</i>	84
5.6.4.	Flutter Google PlayStore App	86

5.6.5.	Azure Triggered Function	87
5.7.	Evolución de los requerimientos no funcionales	87
6.	Gestión del proyecto	88
6.1.	Herramientas para la gestión	88
6.1.1.	JIRA	88
6.2.	Etapa de investigación	88
6.2.1.	Definición de los requerimientos	89
6.2.2.	Investigación de <i>APIs</i> y tecnologías	89
6.2.3.	Comienzo de la documentación	89
6.2.4.	Definición del ciclo de vida y metodología de trabajo	89
6.2.5.	Esfuerzo dedicado	89
6.3.	Etapa de desarrollo	90
6.3.1.	Scrum	90
6.3.2.	Descripción de los <i>sprints</i>	95
6.3.3.	Métricas de gestión	97
6.3.4.	<i>Releases</i>	103
6.4.	Gestión de la comunicación	103
6.4.1.	Comunicación con los interesados	103
6.5.	Gestión de riesgo	104
6.5.1.	Descripción de los tipos de riesgo	105
6.5.2.	Matriz de riesgo (probabilidad e impacto)	106
6.5.3.	Riesgo tipo	106
6.5.4.	Evolución en el tiempo	107
7.	Gestión de calidad	109
7.1.	Aseguramiento de la calidad	109
7.2.	Objetivos de calidad	109
7.2.1.	Objetivos de calidad del producto	109
7.2.2.	Objetivos de calidad del proceso	110
7.2.3.	Justificación de objetivos de calidad	110
7.2.4.	Atributos de calidad	111
7.3.	Planificación de la calidad	111

7.4.	Proceso de calidad.....	113
7.4.1.	Estándares de calidad.....	113
7.4.2.	Revisiones	115
7.4.3.	Actividades de aseguramiento de la calidad	116
7.4.4.	Gestión de incidentes	118
7.4.5.	Métricas.....	120
7.4.5.1.	Métricas de Producto.....	120
7.5.	Conclusiones	127
8.	Gestión de la configuración	128
8.1.	Elementos de configuración de <i>software</i>	128
8.2.	Elección de herramientas	128
8.2.1.	Versionado del <i>software</i> desarrollado.....	128
8.2.2.	Documentación	129
8.3.	Gestión del repositorio Git.....	130
8.4.	Conclusiones	131
9.	Conclusiones	132
9.1.	Estado actual	132
9.2.	Pasos a seguir a nivel de producto	132
9.3.	Evolución y cumplimiento de los objetivos.....	134
9.3.1.	Proyecto	134
9.3.2.	Producto	135
9.3.3.	Académicos.....	139
9.3.4.	Equipo	139
9.4.	Lecciones aprendidas	141
10.	Referencias bibliográficas.....	143
11.	ANEXO.....	145
11.1.	Marco metodológico	145
11.1.1.	Ventajas de Scrum	145
11.2.	Ingeniería de requerimientos.....	145
11.2.1.	I.N.V.E.S.T	145
11.3.	Arquitectura de la solución	145

11.3.1.	Tecnologías	145
11.3.2.	Evolución de los requerimientos no funcionales	147
11.3.3.	Las diez heurísticas de Nielsen	156
11.3.4.	Cumplimiento de las heurísticas de Nielsen	159
11.3.5.	Diagramas de clase (por paquete)	169
11.4.	Gestión del proyecto	178
11.4.1.	Behaviour Driven Development	178
11.4.2.	Gestión del riesgo	178
11.4.3.	Gestión de la comunicación	181
11.4.4.	Documentación de <i>sprint #0</i> (22/4/2024 - 16/6/2024).....	183
11.4.5.	Documentación de <i>sprint #1</i> (17/6/2024 - 1/7/2024).....	185
11.4.6.	Documentación de <i>sprint #2</i> (1/7/2024 - 15/7/2024).....	191
11.4.7.	Documentación de <i>sprint #3</i> (15/7/2024 - 29/7/2024).....	196
11.4.8.	Documentación de <i>sprint #4</i> (29/7/2024 - 12/8/2024).....	201
11.4.9.	Documentación de <i>sprint #5</i> (12/8/2024 - 26/8/2024).....	206
11.4.10.	Documentación de <i>sprint #6</i> (26/8/2024 - 9/9/2024).....	211
11.4.11.	Documentación de <i>sprint #7</i> (9/9/2024 - 23/9/2024).....	216
11.4.12.	Documentación de <i>sprint #8</i> (23/9/2024 - 7/10/2024).....	222
11.4.13.	Documentación de <i>sprint #9</i> (7/10/2024 - 17/10/2024).....	228
11.5.	Plan de calidad	234
11.6.	Atributos de calidad	240
11.7.	Conclusiones	243
11.7.1.	Objetivo "Disfrutar y aprender durante el desarrollo del proyecto"	243

1. Introducción

1.1. Objetivos del proyecto

El objetivo principal de este proyecto es desarrollar e implementar un sistema avanzado de gestión de inventario y análisis de costos para el hotel Tio Tom *All Inclusive*. Este sistema permitirá la automatización y optimización de los procesos de control y gestión de alimentos y bebidas, crucial para la operación de la modalidad *all inclusive* del hotel.

Los diferentes objetivos que determinó el equipo se agrupan por categoría.

1.1.1. Proyecto

1.1.1.1. Desarrollo de un *MVP*

El objetivo se considerará como cumplido únicamente si a la fecha de la entrega administrativa en gestión, el equipo cuenta con un *MVP* que permita realizar las operaciones mínimas/básicas necesarias para la cooperativa del hotel.

KPI: Implementación del *MVP* del sistema (Sí/No)

1.1.1.2. Integración de tecnología avanzada

Este objetivo abarca la incorporación de inteligencia artificial. Se utilizarán algoritmos de *machine learning* para desarrollar modelos predictivos que ayuden en la planificación y gestión de los recursos del hotel, basándose en patrones de consumo históricos y proyecciones de ocupación. Además, el sistema será desarrollado sobre una plataforma que permita la integración continua de nuevas tecnologías y módulos adicionales a medida que vayan surgiendo nuevas necesidades.

KPI: Integración de algoritmos de *machine learning* (Sí/No)

1.1.1.3. Flexibilidad y escalabilidad del sistema

El sistema será diseñado con una arquitectura escalable, permitiendo futuras expansiones o modificaciones sin requerir un gran rediseño. Esta permitirá la capacidad de integrarse y expandirse para incluir otras capacidades que el hotel pueda requerir en un futuro, garantizando una operación más eficiente.

KPI: Capacidad del sistema para integrar módulos adicionales (Sí/No)

1.1.2. Producto

1.1.2.1. Completitud de requerimientos prioritarios/esenciales del producto

Asegurar que todas las funcionalidades de alta o muy alta prioritarias en los requisitos iniciales estén implementadas y funcionen correctamente antes de la entrega del producto. Esto incluye validar cada funcionalidad mediante pruebas exhaustivas.

KPI: El 100% de las funcionalidades prioritarias/detalladas en el documento de requisitos debe estar completado y validado con éxito mediante pruebas funcionales internas antes de la entrega final. (Sí/No)

1.1.2.2. Garantizar la compatibilidad y adaptabilidad del sistema en múltiples plataformas

Asegurar que el sistema funcione correctamente en el entorno *web* tanto en dispositivos móviles como en computadoras, y que la aplicación Flutter sea completamente funcional y se pueda compilar en dispositivos iOS y Android. Se verificará la consistencia de la experiencia de usuario y la funcionalidad entre las diferentes plataformas.

KPI: El sistema *web* se adapta de manera responsive y sin problemas en Chrome en dispositivos móviles y de escritorio, y la aplicación Flutter se puede compilar sin errores críticos en iOS y Android, validado mediante pruebas en diferentes resoluciones de pantalla antes de la entrega del producto. (Sí/No)

1.1.2.3. Calidad y profesionalismo del producto

El sistema será desarrollado siguiendo las mejores prácticas de ingeniería de *software*, asegurando que el producto no solo cumpla con las expectativas funcionales, sino que también ofrezca una interfaz de usuario intuitiva y accesible. Esto asegurará que el personal del hotel pueda adaptarse rápidamente al sistema, minimizando el tiempo de capacitación y maximizando la eficiencia desde el inicio.

KPI: Nivel de satisfacción del usuario con la interfaz y el uso (Escala 1-5)

1.1.3. Académicos

1.1.3.1. Aprendizaje y aplicación de nuevas tecnologías

El proyecto proporcionará al equipo un espacio para profundizar en tecnologías quizás no tan conocidas para ellos, como pueden ser las tecnologías del campo de la inteligencia artificial y *machine learning*. Además, será una oportunidad para aplicar estos conocimientos en un contexto

real y desafiante, lo cual fortalecerá sus habilidades y preparación para futuros desafíos profesionales.

KPI: Adopción de tecnologías nuevas y desafiantes (o escasamente utilizadas por el equipo) como Python (Sí/No)

1.1.3.2. Aplicación práctica de conocimientos académicos

El desarrollo del sistema permitirá al equipo aplicar y combinar una variedad de conocimientos y técnicas aprendidas durante la carrera. Esto incluye desde la programación y el diseño de un *software*, hasta la gestión de proyectos, proporcionando una experiencia a través de todo el ciclo del desarrollo de un *software*, lo cual refleja las exigencias del mundo real.

KPI: Cantidad de técnicas académicas aplicadas en el proyecto

1.1.4. Equipo

1.1.4.1. Compromiso con el equipo

El éxito del proyecto se basa en un trabajo en equipo efectivo y un entendimiento mutuo entre los integrantes. Para eso es necesario que los integrantes puedan acudir a las reuniones realizadas. Estas reuniones, son instancias claves para compartir avances, dudas, preguntas, logros, etc.

KPI: El 100% de los integrantes acudió a todas las reuniones clave (Scrum *daily, planning, review* y *retrospective*). (Sí/No)

1.1.4.2. Creación de un emprendimiento tecnológico

Explorar la posibilidad de transformar el proyecto en un emprendimiento comercial viable es una motivación adicional para el equipo, ya que puede desencadenar el surgimiento de un producto comercial. Esta posibilidad no solo proporciona un potencial retorno económico a medio plazo, sino que también permite aplicar y expandir las soluciones desarrolladas a un mercado más amplio, adaptándolas a las necesidades de otros clientes en la industria.

KPI: Viabilidad del proyecto como emprendimiento (Sí/No)

1.1.4.3. Disfrutar y aprender durante el desarrollo del proyecto

Más allá de los objetivos técnicos y académicos, el equipo valora la importancia de disfrutar el proceso de aprendizaje y desarrollo. Aprovechar al máximo las oportunidades que brinda este proyecto para crecer personal y profesionalmente, es fundamental para el enriquecimiento de cada miembro del equipo.

KPI: Nivel de satisfacción del equipo con el proceso de desarrollo (Escala 1-5)

2. Problema y su solución

2.1. Introducción

Cuando el Hotel Tío Tom *All Inclusive* comenzó a operar, lo hizo bajo un enfoque más tradicional, conocido como "breakfast only". Esto significa que, en la tarifa abonada por los huéspedes, únicamente se incluía el desayuno, sin cubrir otras comidas del día. Esta modalidad permitió al hotel ofrecer un servicio sencillo y directo, adaptado a las necesidades de aquellos huéspedes que preferían organizar sus propios almuerzos y cenas fuera del hotel.

No fue sino hasta el año 2021 que el hotel implementó un cambio significativo en su modelo de servicio, pasando de "breakfast only" a "*all inclusive*". Este cambio marcó un hito en la historia del hotel, ya que se convirtió en el primer establecimiento en la costa este de Uruguay en operar bajo esta modalidad. La transformación no solo representó una evolución en la oferta de servicios, sino que también planteó un desafío considerable para la gestión y administración del hotel.

Con el nuevo modelo de "*all inclusive*", la tarifa no solo incluía el desayuno, sino también las cuatro comidas principales del día: desayuno, almuerzo, merienda y cena. Además, los huéspedes obtuvieron acceso a un snack bar disponible durante todo el día, donde podían disfrutar de una variedad de alimentos como hamburguesas, panchos, papas fritas y otros aperitivos. Este cambio radical en la oferta gastronómica implicó un esfuerzo significativo por parte del hotel, no solo en términos logísticos, sino también en la forma en que se gestionaba la experiencia de los huéspedes.

La implementación del "*all inclusive*" supuso una redefinición del enfoque del hotel, que pasó de ofrecer una simple estancia con desayuno a brindar una experiencia integral donde los huéspedes no necesitaban preocuparse por las comidas durante su estadía. Esto no solo aumentó el nivel de satisfacción de los clientes, sino que también posicionó al hotel como una opción líder en el turismo de la zona este del país, atrayendo a un nuevo tipo de huésped que buscaba comodidad y un servicio más completo durante sus vacaciones.

2.2. Problema

Cuando el Hotel Tío Tom decidió pasar de la modalidad "Breakfast Included" a "*All Inclusive*", el control de costos en el sector de Alimentos y Bebidas dejó de ser un tema menor y se convirtió en uno de los principales desafíos operativos. La transición implicó un cambio fundamental en la forma en que los huéspedes interactuaban con los servicios de comida, ya que ahora contaban con una mayor variedad y libertad a la hora de elegir qué consumir, cuánto consumir y cuándo hacerlo. Esta variabilidad en el consumo introdujo una nueva complejidad para la gestión del inventario y la planificación de los insumos.

Para el equipo de cocina, adaptarse a este nuevo modelo fue especialmente complicado. Aunque podían apoyarse en los registros históricos del hotel, donde el promedio diario de huéspedes era

de 146 y el número aproximado de comensales por semana alcanzaba los 1025, prever con exactitud las necesidades diarias de alimentos y bebidas resultaba un reto. La dificultad radicaba en que no solo se trataba de mantener un equilibrio adecuado entre demanda y oferta, sino que también era necesario gestionar la preparación de platos sin caer en excesos que generaran desperdicios, ni en carencias que pudieran afectar la experiencia del huésped.

Actualmente, el hotel maneja entre 500 y 700 variedades diferentes de productos relacionados con la comida y bebida, una cantidad significativamente mayor comparada con los pocos productos que se gestionaban bajo la modalidad "Breakfast Included". Este aumento drástico en la variedad de insumos también complicó la logística de almacenamiento y la relación con los proveedores, que pasó de ser relativamente simple a requerir la coordinación con un mayor número de ellos. Esta complejidad adicional en la cadena de suministros incrementó la necesidad de realizar una gestión más detallada y eficiente, tanto en términos de recepción como de control de stock.

Además, el hotel se vio obligado a aumentar el personal en el área de cocina y atención al cliente para poder sostener el nuevo ritmo de operación. Aparte de esta expansión, se contrató un empleado dedicado exclusivamente a la gestión de inventarios y costos, encargado de controlar el flujo de insumos, asegurar que los recursos se utilizaran eficientemente y gestionar las facturas emitidas por los proveedores.

Por último, la variabilidad en el consumo de los huéspedes fue otro factor problemático. A diferencia del desayuno, que tiene un consumo más predecible, las demás comidas y snacks presentaron un comportamiento fluctuante, haciendo más difícil planificar las compras y la preparación de los alimentos. Algunos huéspedes aprovechaban al máximo la oferta, mientras que otros consumían menos de lo esperado, lo que resultaba en desabastecimiento en ciertos momentos o en el desperdicio de productos que no se utilizaban.

En resumen, la gestión de Alimentos y Bebidas se convirtió en un reto considerable para el hotel. Pasaron de manejar una operación simple y predecible a enfrentarse con una mayor diversidad de productos, proveedores y empleados, todo bajo un consumo variable y difícil de anticipar. Esta transformación hizo que el control de costos y la planificación de inventarios se convirtieran en tareas esenciales y más desafiantes para el éxito del hotel.

2.3.Solución

Francisco Cabanillas, integrante del equipo del proyecto, desempeñó diversas funciones dentro del hotel entre las temporadas 2019/20 y 2022/23, lo que le permitió experimentar de primera mano los problemas generados por el cambio a la modalidad "*All Inclusive*". Consciente de las dificultades que enfrentaba el hotel en la gestión de alimentos y bebidas, decidió aplicar sus conocimientos en el desarrollo de una solución tecnológica. Aprovechó las bases teóricas y prácticas que había adquirido recientemente desarrollando el obligatorio correspondiente a la

materia Diseño de Aplicaciones 1 y se puso manos a la obra para diseñar un sistema que pudiera dar respuesta a la situación.

Esta primera versión de la solución fue un sistema básico para el análisis de costos del "*All Inclusive*". Se trataba de una aplicación de escritorio para Windows, desarrollada apresuradamente en .NET Framework y basada en el esqueleto de una aplicación de Finanzas Personales que Francisco había creado anteriormente, junto a su compañero de clases y también estudiante de Universidad ORT, Alfonso Irazábal. La base de datos era local y estaba instalada en la laptop del analista de costos, lo que limitaba su accesibilidad y escalabilidad. Además, el sistema requería la carga manual de las facturas, lo que hacía necesario que el analista estuviera dedicado a tiempo completo a esta tarea, y los reportes también debían extraerse manualmente.

A pesar de sus limitaciones, el sistema proporcionaba una estructura inicial para la gestión de costos. El relevamiento de stock se realizaba semanalmente en conjunto con el chef, lo que permitía obtener una mejor visibilidad de los insumos utilizados. Aunque el desarrollo se llevaba a cabo semana a semana, con un conocimiento limitado del funcionamiento de la modalidad "*All Inclusive*", esta solución rudimentaria sirvió como punto de partida.

A partir de esta base, el equipo trabajó para escalar la solución, integrar mejoras y ajustarse a los nuevos requerimientos del cliente. La nueva solución fue desarrollada desde cero, pero aprovechando el proyecto inicial como fuente de aprendizaje y experiencia.

Esta nueva versión estará compuesta por varios productos destinados a optimizar la gestión del hotel. En primer lugar, se desarrollará un *frontend web* en Angular, cuyo principal objetivo será servir como panel de administración, permitiendo además la visualización de reportes y análisis detallados de los costos.

Adicionalmente, se implementará un *frontend* en Flutter para ser utilizado en tabletas dentro del almacén donde actualmente se guardan los productos. Este sistema permitirá registrar de manera eficiente los egresos e ingresos de productos al inventario. Además, un empleado dispondrá de una tableta para contabilizar la cantidad de huéspedes por cada comida del día, así como las habitaciones a las que pertenecen.

En cuanto al *backend*, este será responsable de gestionar todo lo relacionado con el inventario, los productos y los proveedores. Una de las características clave será la automatización de la carga de facturas emitidas al hotel por parte de los proveedores de alimentos y bebidas, lo que reducirá significativamente la carga de trabajo manual y los errores.

Por último, la solución contará con un módulo de *machine learning* que permitirá realizar predicciones más precisas sobre la variabilidad en el consumo, los costos y otros factores asociados. Este módulo proporcionará al hotel una herramienta avanzada para mejorar la planificación y la toma de decisiones, optimizando así su operación bajo la modalidad "*All Inclusive*".

3. Marco metodológico

3.1. Ciclo de vida

Dadas las características del proyecto y el contexto del cliente, el equipo optó por un enfoque de desarrollo incremental [14]. Este ciclo de vida es ideal para adaptarse a las necesidades cambiantes y complejas de un sistema de gestión de inventario y costos.

Los ciclos incrementales son fundamentales en este enfoque, ya que permiten la repetición de actividades (iteraciones) en fases específicas para maximizar el entendimiento del producto por parte del equipo. El objetivo es que al final de cada iteración, se cuente con un entregable funcional que refleje una parte del sistema y que cumpla con las consignas/objetivos de la iteración. Esto permite validar funcionalidades y hacer ajustes antes de proceder a la siguiente fase. Ese entregable puede ser interno para el equipo y oculto para el cliente, debido a que pueden tratarse de avances que el cliente no puede percibir (por ejemplo, funcionalidades de la *API*, pero no en la *web*).

La flexibilidad del ciclo de vida incremental facilita la incorporación de cambios en los requisitos y necesidades del proyecto, lo cual es crucial en un entorno que debe adaptarse constantemente a variaciones de demanda y oferta turística. Este modelo de ciclo de vida proporciona la estructura necesaria para manejar de manera efectiva los cambios y asegurar que el producto final cumpla con los requisitos identificados en cada etapa del proceso.

Por ejemplo, si se identifica la necesidad de mejorar el módulo de gestión de facturas, se puede planificar su implementación en una iteración posterior/diferente sin interrumpir el desarrollo en curso de otras partes del sistema.

Al final de cada iteración, se evalúa el entregable obtenido, lo que permite no solo verificar la funcionalidad sino también la alineación con los requerimientos establecidos. Esto es vital para asegurar que el sistema final realmente permita al hotel poder optimizar la gestión de inventarios y costos, mejorando así la rentabilidad y eficiencia.

El uso de un enfoque incremental también facilita la gestión de riesgos, permitiendo identificar y resolver problemas técnicos o de diseño en fases tempranas del desarrollo, antes de que estos puedan impactar significativamente el progreso del proyecto o la calidad del producto final.

Este ciclo de vida no solo le asegura al equipo un desarrollo ágil y orientado a resultados, sino que también permite responder de manera efectiva a los cambios, maximizando así las oportunidades de éxito del proyecto y la satisfacción del cliente.

Al no ser un proyecto de requerimientos cambiantes, o que los mismos van surgiendo a medida que se desarrolla, el ciclo de vida evolutivo fue descartado [15].

3.2. Metodologías ágiles

3.2.1. Scrum

Para el desarrollo del sistema, el equipo optó por un enfoque de desarrollo incremental utilizando una adaptación de la metodología ágil Scrum. Este ciclo de vida es ideal en contextos donde la respuesta rápida a posibles cambios es crucial. No es el caso del equipo, ya que se cuentan con los requerimientos ya establecidos. De igual manera, Scrum otorga la flexibilidad necesaria para afrontar cualquier cambio repentino en los requerimientos, en el caso que se produzca alguna transformación.

Por esa razón el equipo considera que no es una metodología cien por ciento ágil, sino que se habla de una metodología del tipo híbrida.

Como la metodología seleccionada trabaja por medio de iteraciones llamadas *sprints*, las mismas obligan al equipo a producir un incremento al finalizar dicho *sprint*.

El porqué del término "...adaptación de la metodología ágil Scrum", radica en que el equipo no implementó Scrum puro, debido a que no se utilizaron todos los artefactos y eventos de la metodología de una manera cien por ciento formal. Como, por ejemplo, el Scrum *daily*. Debido a las agendas de cada integrante del equipo, se dificultó la realización de una reunión diaria de lunes a viernes. Para contrarrestar esta problemática, el equipo fijó los jueves, como el día adecuado para que acontezcan estas reuniones.

El equipo tiene experiencia previa con Scrum, lo cual es de gran valor para enfrentar los desafíos del proyecto. Esta experiencia permite implementar las prácticas de Scrum de manera efectiva y adaptarlas a las necesidades específicas del cliente.

En el anexo [Ventajas de Scrum](#), se pueden visualizar las ventajas.

3.2.2. Comparativa Scrum vs Kanban

La elección de Scrum sobre otras metodologías ágiles como Kanban, se basó en varios factores específicos del proyecto. Primero, la naturaleza iterativa y estructurada de Scrum, con sus ciclos definidos de *sprints* y *reviews*, se alinea bien con la necesidad de obtener resultados medibles en intervalos regulares. Esto es crucial para un proyecto donde las entregas incrementales son fundamentales para el éxito de este tipo de proyecto en particular.

Además, Scrum promueve una colaboración continua entre todos los miembros del equipo y el cliente, facilitando una comunicación abierta y un entendimiento común de los objetivos del proyecto. Esto asegura que se puede ver el progreso en tiempo real, lo cual es vital para mantener el proyecto en curso y adaptarse rápidamente a cualquier cambio o desafío.

3.2.3. Descartar Kanban

En cuanto a Kanban, aunque es muy recomendado para gestionar el flujo de trabajo y es altamente flexible, no impone límites de tiempo estrictos como los *sprints* en Scrum. Esta característica podría haber resultado en menos presión para cumplir con entregas específicas en plazos estrictos, lo cual podría ser menos ideal para un proyecto con requisitos y expectativas claros y con un calendario ajustado. Además, el equipo directamente no cuenta con experiencia en Kanban, y además Scrum ofrece un marco más familiar y controlado para gestionar las complejidades del proyecto.

Por último, Scrum fue seleccionado por su capacidad para facilitar una mejor estimación y planificación a través de su enfoque en la priorización del *backlog* y la revisión continua del progreso. Esto no solo ayuda a mejorar la eficiencia del equipo, sino que también asegura que el proyecto permanezca en camino hacia el cumplimiento de los objetivos finales, maximizando el retorno de inversión para el cliente y minimizando los riesgos asociados con el desarrollo del producto.

3.3. Etapas del proyecto

El desarrollo del sistema de gestión de inventario y análisis de costos se estructuró en tres etapas principales, cada una diseñada para abordar los distintos aspectos del proyecto de forma secuencial y efectiva.

3.3.1. Investigación y análisis

La primera fase del proyecto se centró en diferentes tareas como:

- Definición de los requerimientos del *software*
- Inicio de las reuniones con la tutora Helena Garbarino.
- Comienzo de la documentación.
- Definición del ciclo de vida y metodología de trabajo.
- Investigación de la *API* de FacturaLista y DGI, la cual es una integración con terceros que si o si necesita el *software*.
- Elección de tecnologías.
- *Sprint 0* (en el cual no se desarrolló *software*).

3.3.2. Desarrollo y pruebas

Tras concluir la fase inicial, se procedió a la etapa de desarrollo del producto. Utilizando la metodología ágil Scrum, adaptada a las necesidades y tiempos del equipo, se logró una mayor flexibilidad y respuesta rápida ante los desafíos. Durante esta fase, también se puso especial

atención en la validación y las pruebas del sistema para garantizar su funcionalidad y calidad, asegurando que cada incremento cumpliera con los estándares de calidad y requerimientos existentes.

3.3.3. Fiscalización y mejora continua del producto

La etapa final se dedicó a la implementación de todas las mejoras posibles previo a la fecha de finalización del proyecto.

Esta fase fue esencial para refinar el producto final previo a la entrega en gestión, asegurando también una documentación detallada del msmo. El objetivo fue consolidar todos los aspectos del proyecto, desde la funcionalidad hasta la interfaz de usuario, para garantizar una solución robusta que respondiera a los requerimientos del cliente.

3.4. Roles

Durante la primera reunión con la tutora del equipo, se le asignó al equipo la tarea de asignar los roles de los integrantes del equipo. Esta distribución de roles se basa en los niveles de experiencia de cada miembro, las preferencias personales y las áreas en las que cada uno demostró una mayor habilidad.

Este proceso de definición de roles fue muy importante para la gestión del equipo durante el proyecto. Al alinear los roles con las habilidades y preferencias individuales, se facilita la colaboración. Esta estrategia aseguró que cada área crítica del proyecto fuera liderada por alguien con el conocimiento y la motivación necesarios para generar un mayor progreso en el proyecto.

Project manager: Francisco Cabanillas

Aunque el rol de *project manager* no es formal dentro del marco de Scrum, Francisco Cabanillas asume esta posición estratégica que abarca el proyecto en su totalidad. Como *project manager*, Francisco se encarga de coordinar y supervisar el desarrollo general del proyecto, asegurando que todas las partes avancen coherentemente hacia los objetivos establecidos. Su gestión incluye el manejo del cronograma, los recursos y las comunicaciones del equipo, lo cual es crucial para el flujo eficiente del trabajo y la resolución de conflictos o retrasos. Además, Francisco actúa como el principal intermediario entre el equipo de desarrollo y el *product owner*, facilitando un canal claro y efectivo para el reporte de avances y la retroalimentación continua. Esta interacción asegura que las expectativas del *product owner* se mantengan alineadas con el progreso del proyecto, y que cualquier ajuste necesario sea implementado de manera oportuna.

Ingeniero en requerimientos: Francisco Cabanillas

Francisco Cabanillas también se desempeña como ingeniero en requerimientos, una posición crítica que implica identificar, documentar y validar los requerimientos del proyecto en

colaboración con el cliente. Su trabajo es esencial para asegurar que los requerimientos tanto funcionales como no funcionales se integren adecuadamente en la arquitectura del sistema. Francisco también se encarga de monitorear la trazabilidad de estos requerimientos a lo largo del proyecto y facilita la comunicación entre el equipo técnico y los clientes o usuarios finales para asegurar una clara comprensión y alineación de los objetivos del proyecto.

Además, es el encargado de identificar y gestionar los riesgos potenciales que podrían afectar la culminación exitosa del proyecto.

Arquitecto de software: Martín Sosa

En su rol como arquitecto de infraestructura, Martín asume la responsabilidad de diseñar y proponer soluciones de infraestructura tecnológica que soporten los requerimientos establecidos para el sistema. Su trabajo debe garantizar la escalabilidad, seguridad y rendimiento de la infraestructura tecnológica, supervisando la implementación y realizando ajustes necesarios, de acuerdo con las necesidades del proyecto. Además, colabora en la integración de los distintos componentes y servicios que conforman el sistema, asegurando que se tenga una infraestructura eficiente.

Encargado de gestión: Martín Sosa

Martín Sosa ocupa la posición de encargado de gestión, donde sus principales tareas son: Procurar el cumplimiento de las prácticas de Scrum, no excederse de los tiempos acordados (en lo posible, asegurar los entregables dentro del proyecto y coordinar las tareas necesarias para mantener el equipo operativo y eficiente.

Martín juega un papel clave en facilitar la comunicación interna y asegurar que todos los miembros del equipo cuenten con las herramientas e información necesarias para su trabajo.

Encargado de la gestión de calidad: Andrés Quintero

Andrés Quintero, como encargado de gestionar la calidad, define y supervisa los estándares de calidad del proyecto para asegurar que el sistema cumpla con estos criterios antes de su lanzamiento. Andrés supervisa la implementación de mejores prácticas de codificación y revisión de código, y documenta los resultados para mantener un control de calidad riguroso. Su trabajo garantiza que el proyecto alcance los niveles de calidad requeridos y satisfaga las expectativas del cliente.

3.5. Cronograma del proyecto

Como anteriormente el equipo comentó, el cronograma del proyecto se estructura en *sprints*. La siguiente imagen (

Ilustración 1) muestra los diferentes hitos a lo largo de los *sprints*:

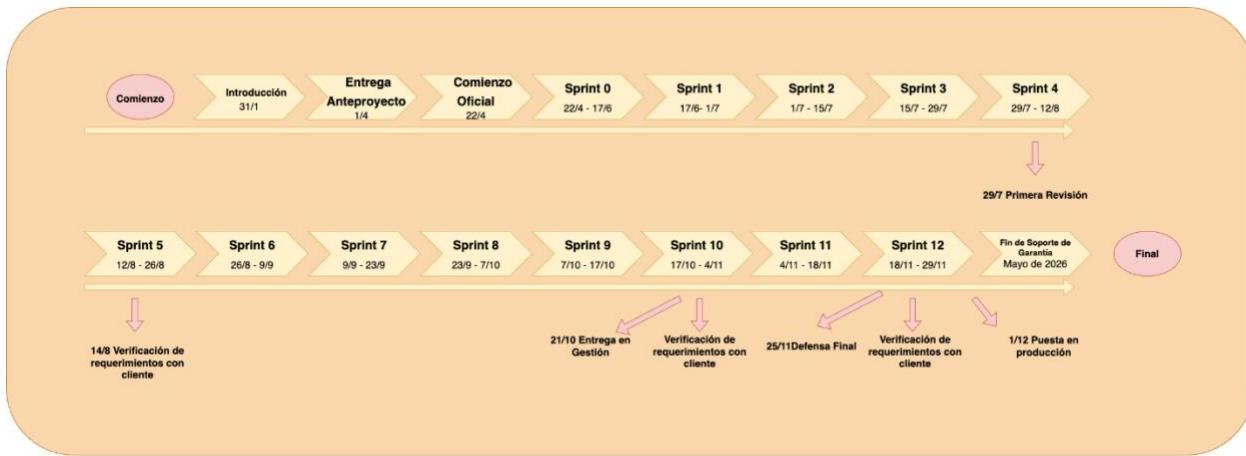


Ilustración 1 - Cronograma del proyecto

A continuación, se describen los hitos clave del proyecto.

3.5.1. Verificación de requerimientos con el cliente

Una vez transcurridos 5 *sprints* de desarrollo, el equipo logró un avance visual como para ser mostrado al cliente. Se procedió a agendar una videollamada, para hacer una primera demo de los requerimientos que fueron materializados en la interfaz gráfica, tanto del panel *web*, como de la aplicación móvil.

Respecto a la aplicación *web*, se mostró el flujo del *login/autenticación* de la aplicación *web*, para luego llegar al menú principal de la misma (sin funcionalidad aún, solo el maquetado) y mostrar la vista genérica de los CRUD's.

Por otro lado, en la aplicación móvil fueron mostrados los siguientes requerimientos: *login, logout, registro de movimientos de inventario, eliminación de movimientos de inventario y listado de movimientos de inventario*.

El cliente mostró su absoluta conformidad con lo acontecido, tanto en *web* como móvil. No se evidenciaron críticas. Hizo hincapié en un requerimiento importante (aún no implementado, pero ya conocido), el cual consta de dejar registro de la cantidad de comensales por turno (desayuno, almuerzo, merienda y cena) del hotel, para que el equipo lo tuviese en consideración (para la *app* móvil). De esta manera, ellos pueden conocer, por ejemplo, cuántos comensales hubo en el almuerzo del 2 de enero del 2025.

Al momento de la primera reunión de verificación de requerimientos, la aplicación móvil *Flutter*, se encontraba en un 50%, mientras que la *web* se encontraba en su etapa inicial.

En lo que corresponde a verificación de requerimientos y usabilidad desde el punto de la interacción del cliente con el producto, el equipo tiene pensado realizar dos próximas sesiones durante los *sprints* 10 (desde 17/10/2024 hasta 4/11/2024) y 11 (desde 4/11/2024 hasta 18/11/2024), posterior a la entrega del documento en gestión.

3.5.2. Entrega del anteproyecto

El anteproyecto se entregó el 22 de abril de 2024. Este hito es crucial ya que estableció las bases del proyecto y definió en parte, los objetivos y requerimientos específicos.

3.5.3. Revisión del proyecto

El 29 de julio de 2024 se realizó la primera y única revisión del proyecto con Marcelo Cagnani.

3.5.4. Revisión de la arquitectura

El equipo contó con el apoyo del profesor Gastón Mousqués con quien se realizó una reunión para presentarle la arquitectura del sistema.

3.5.5. Entrega administrativa en el portal de gestión

El 21 de octubre de 2024, se realiza la entrega de la documentación gestión. Para entonces, se habrán completado varias iteraciones enfocadas en la implementación de las funcionalidades requeridas (el ciclo de vida del desarrollo no finaliza con esta entrega).

3.5.6. Defensa

Programada para los días 25 al 28 de noviembre de 2024, la defensa es un hito donde el equipo presentará el proyecto finalizado ante el comité evaluador. Este evento se llevará a cabo después de completar múltiples *sprints* enfocados en pulir detalles, corregir errores y optimizar el sistema basado en las pruebas iniciales y el *feedback* recibido. La defensa será una oportunidad para demostrar la efectividad de la solución y la alineación con los requerimientos del cliente.

3.5.7. Puesta en producción

El 1 de diciembre de 2024, el sistema será puesto en producción, marcando la culminación de los *sprints* dedicados a asegurar la estabilidad y la funcionalidad del *software* en un entorno real. Este hito es fundamental, ya que implica que el sistema estará completamente operativo y listo para su uso diario por parte del hotel, mejorando significativamente la eficiencia en la gestión de costos y stock.

3.5.8. Fin de soporte de garantía

El soporte de garantía finalizará a finales de abril de 2026. Este período permitirá al equipo asegurar que el sistema funcione sin inconvenientes y que todas las funcionalidades implementadas estén operativas y optimizadas.

En total, el proyecto abarca nueve *sprints* hasta la entrega de gestión, más varios *sprints* adicionales hasta la puesta en producción y el fin del soporte de garantía. Este enfoque iterativo asegura una adaptación continua a los requerimientos del cliente y una mejora constante del producto a lo largo del ciclo de vida del proyecto.

4. Ingeniería de requerimientos

4.1. Obtención y análisis de requerimientos

El relevamiento de requerimientos es una fase crítica en el desarrollo de software, ya que implica la recopilación de información sobre las necesidades y expectativas de los usuarios y *stakeholders*. Para realizar este proceso eficazmente, se utilizan herramientas que facilitan la obtención y organización de la información.

4.1.1. Herramientas para el relevamiento

Este relevamiento se realizó en su mayoría previo a la iniciación del proyecto para la universidad en el verano de 2021, cuando el hotel en cuestión se embarcó en su primera instancia de formato All Inclusive. Esa etapa realizada antes no tuvo un proceso formal que incluyera encuestas, mapas mentales u otros debido al apuro que los altos cargos conocían los gastos realizados, para saber si era redituible el formato o si se excedían.

4.1.1.1. Observación directa

La observación directa en el ámbito laboral es una técnica eficaz para la recopilación de requerimientos de software, ya que permite a los analistas observar cómo los empleados interactúan con los sistemas y procesos actuales en su entorno de trabajo. Al estar inmersos en el contexto laboral, los observadores pueden identificar patrones de comportamiento, detectar problemas y descubrir necesidades que los usuarios pueden no haber expresado durante entrevistas o encuestas. Este ciclo de vida proporciona información valiosa sobre la dinámica del trabajo, los flujos de información y las interacciones entre los empleados y la tecnología, empatizando con los actores involucrados en los procesos, lo que facilita la comprensión de sus verdaderas necesidades.

Esta fue la técnica **utilizada** en el desarrollo de la primera versión **utilizada** en el pasado por Francisco. En el análisis de costos, él entendió primero cómo eran los procesos del sector de Alimentos y Bebidas y el sector administrativo y de finanzas con preguntas como: "¿Qué días se realiza el stock en la cocina?", 2. "¿Qué diferencia tienen las unidades en las facturas contra la realidad?", 3. "¿Por dónde llegan las facturas?", 4. "¿Qué número quieren ver los gerentes?", y lo más importante, 5. "¿Cómo lograr estos cálculos diaria o semanalmente?"

A partir de lo investigado encontró que:

- 1- Se realizaban los stocks los sábados, demorando unas 4 horas en contar y pesar todos los productos, precisando también así que todos los datos de las facturas estuvieran cargados para este día.

- 2- Dependiendo del proveedor, varía la nomenclatura del mismo producto para representar distintas unidades, no siempre siendo exacta la relación entre ellas. Por ejemplo: ‘BANANA ecuador IMP KG’ / ‘BANANA ecuador IMP CAJA’ (Ilustración 2)

5,00	BANANA Ecuador IMP KG	79,00	0,00	395,00
2,00	BANANA Ecuador IMP CAJA	1.540,00	0,00	3.080,00

Ilustración 2 - Detalle de factura con diferentes unidades

Las cajas suponen tener entre 15 kg y 20kg de producto aproximado dependiendo de cual se trate, las bolsas tienen 10 kg aproximadamente.

- 3- Las facturas que se emitían hoy en día al hotel, y al parador hasta el año pasado, se pueden encontrar en el proveedor electrónico de ellos ‘Factura Lista’, siempre y cuando el proveedor tenga facturación electrónica. Estas se descargaban en formato pdf y eran diferenciadas entre hotel y parador por el encargado de alimentos y bebidas (lo cual ya no es necesario desde 2024 por su separación de razones sociales). También había proveedores con facturación manual, por lo que el chef debía guardar estas facturas y de perderse habría que pedir al proveedor una imagen de su copia.
- 4- Cuando se presupuestó la idea, se estimó un consumo de 25 dólares por huésped por día. Para lograr un cálculo comparativo, se podía dividir el gasto total (de todos los tipos de producto) por la suma de huéspedes de cada día. Ejemplo: Un promedio de 90 huéspedes por día son 630 huéspedes en 7 días. Calculando todo lo comprado en esos 7 días, dividido por estos 630 huéspedes que fueron alimentados, se obtiene las compras por huésped por día.
- 5- A partir de lo investigado y con las últimas cuatro respuestas se dimensiona la magnitud del problema principal. Excel como herramienta no iba a ser viable por la ineficiencia y el sesgo que presentaría. ¿Por qué? El problema de realizar estos cálculos solo con los totales de las facturas no muestra lo que consumieron esos huéspedes, sino que se infla durante una semana en la que varios productos se terminaron y se tuviera que hacer un encargo grande, como la semana previa al encargo grande sería la de menor compras.

4.1.1.2. Prototipado y uso

Un proceso de prototipado y uso de un sistema empieza con la creación de un prototipo básico del software, con las funcionalidades identificadas por la observación directa y otros métodos de recolección de requerimientos. Este prototipo permite a los usuarios interactuar con el sistema en un entorno controlado, brindando una representación visual y funcional de cómo será el producto final.

Una vez que el prototipo está en funcionamiento, se invita a los empleados a probarlo. Durante esta fase, se recopilan comentarios sobre su experiencia, usabilidad y si el sistema satisface sus necesidades.

Para este caso el empleado que usaba el sistema era el mismo desarrollador del prototipo: Francisco. Esto agilizo el proceso de recopilación viendo él mismo las falencias u oportunidades que habían de mejora. Por supuesto que toda sugerencia era bienvenida, especialmente por el chef con el que se tenía un trato directo en el uso del prototipo.

A medida que se obtenían nuevas ideas y sugerencias, el prototipo se ajustaba y refinaba en ciclos iterativos. Este enfoque no solo mejora el diseño del sistema, sino que también permite adaptarse a las necesidades cambiantes del usuario y del negocio. Así, el prototipado se convierte en una herramienta dinámica para descubrir continuamente la información necesaria y optimizar el software de acuerdo con las expectativas de los usuarios.

4.1.1.3. Reunión virtual con el cliente

Conociendo las falencias que había en el hotel con respecto a este sector cuando se presenta la oportunidad de realizar el proyecto de fin de carrera, la primera idea fue ofrecerle al hotel trabajar en conjunto para solucionar estos problemas con los que contaban.

Se realizó una videoconferencia entre Francisco, el dueño Roberto Planas y el encargado Maximiliano Correa donde se conversó por arriba sobre el prototipo que se había utilizado en previos años y lo que se podría lograr usándolo como base para avanzar en esta integración tecnológica en un hotel tan antiguo fundado por el padre de Roberto.

La mayor duda que presentó el dueño fue por qué el servicio de facturación para restaurantes que utilizaban en el parador no podría utilizarse para esta situación. La situación con ese sistema era la manera en que estaba implementado, donde hay un manejo de stock, pero este disminuye a partir de las cuentas que se le generan a las mesas. De cada plato se configura en el sistema cuando consumía de cada producto para realizarse y a partir de esto el sistema sabe las cantidades a restar del stock cuando se sirve ese plato en cualquier cuenta en la jornada. Sabiendo esto vemos la dificultad que presenta el uso de ese sistema en un servicio donde cada huésped elige con mano propia el tamaño de su plato, con que completa el mismo, la cantidad de repeticiones que desea y todo realizado con mínima intervención de un profesional del hotel.

Tanto Maximiliano como Francisco tenían clara esta diferencia en ese instante y pudieron clarificarla detalladamente para poder así continuar.

4.1.1.3.1. Entrevista presencial con el cliente

El proceso de entrevista al cliente es crucial para recopilar y revisar requerimientos en el desarrollo de software, ya que permite obtener información directa sobre las necesidades y expectativas de

los usuarios finales. Durante estas entrevistas, se utilizan preguntas abiertas en un ambiente de confianza, lo que facilita la expresión de opiniones y la identificación de problemas con el sistema actual. El analista escucha atentamente, toma notas y realiza preguntas de seguimiento para clarificar respuestas. Este enfoque no solo genera un conjunto de requerimientos más claro, sino que también establece una relación de colaboración fundamental para el éxito del proyecto.

Esta ocasión también aportó a confirmar los requerimientos ya relevados en otras instancias y poder mejorarlo o refinarlos.

En febrero todo el equipo fue invitado a pasar una noche en el hotel para reunirse en la mañana con Maximiliano Correa. El motivo de esta instancia era repasar y adaptar los requerimientos relevados para el prototipo con los nuevos procesos implementados en estos nuevos años. Por ejemplo, la separación de razones sociales entre el hotel y parador, la obligación de los comercios desde 2024 a facturar solo por medio electrónico y la integración del nuevo container para almacenar los productos. Se recorrieron las instalaciones desde adentro, conversando superficialmente sobre las dinámicas gastronómicas con las que cuentan, conociendo referentes como el nuevo encargado de alimentos y bebidas, previamente encargado de barra, Augusto Gavernet.

Estas dinámicas gastronómicas constan de noches temáticas (como la noche mexicana con tacos), algunas con *show cooking*, en las que siempre se puede elegir sobre una variedad de proteínas, carbohidratos, postres dulces o salados, así como un bar con bebidas para todos los gustos.

También se nos comentó las dificultades con las que contaron para poder analizar los costos en la última temporada que optaron por no usar el prototipo. Hicieron el intento de usar excel solo considerando los valores totales de las facturas.

Lamentablemente el dueño no pudo estar presente en esta instancia, pero mostraba estar de acuerdo con la idea en su mayoría del tiempo todavía con algunas dudas respecto a la diferencia que se obtendría con el programa de restaurante antes mencionado, pero confiando en Maximiliano (como el actual encargado) y Francisco (como su empleado con el que presenta un cierto nivel de confianza).

4.1.2. Conclusión

A partir de estas instancias y la información recopilada, pudimos reconocer a los principales interesados, siendo estos: el dueño, el chef, el encargado de alimentos y bebidas, y el analista de costos. Todos afirman la falta de integración con la tecnología en ese sector del hotel y visualizan los beneficios que podría tener la implementación de un sistema que contabilice los costos y pueda llevar un stock en tiempo real de los productos.

Esta necesidad de modernización resalta la importancia de adoptar herramientas tecnológicas que no solo optimicen la gestión de inventarios y costos, sino que también mejoren la toma de decisiones estratégicas. La implementación de un sistema integral permitiría a los interesados acceder a datos actualizados sobre el uso de insumos, facilitando la planificación de compras y reduciendo el desperdicio. Además, una solución tecnológica podría mejorar la comunicación entre los diferentes departamentos, asegurando que todos trabajen con la misma información y en la misma dirección.

En resumen, la integración de tecnología en la gestión de alimentos y bebidas del hotel no solo es deseable, sino que se ha convertido en una necesidad imperante para mejorar la eficiencia operativa y maximizar la rentabilidad del negocio.

4.2. Especificación de requerimientos

La etapa de Especificación de Requerimientos es un proceso integral que implica la recopilación, análisis, documentación, revisión y aprobación de los requerimientos, asegurando que se establezcan bases sólidas para el desarrollo del proyecto.

4.2.1. Prototipo "Finanzas Tío Tom"

El prototipo como se comentó no tenía esperanzas de terminar siendo un sistema de análisis de costos, sino que comenzó como una aplicación de escritorio para finanzas personales. Esto dificultaba mucho su modificación y extensibilidad, especialmente con el poco tiempo con el que se contaba. Este prototipo contaba con Proveedores, Sectores (de producto), Productos y Gastos, siendo esta la estructura base desde la que partimos.

Esta aplicación era local y únicamente accesible por una instancia a menos que se invirtiera en descentralizar su base de datos.

Nombre	IVA	Sector
ADOBIO	22	Almacen

Ilustración 3 - Prototipo: Vista de ingreso de gastos

El ingreso y trazabilidad de los gastos (Ilustración 3) de cada producto era uno de los mayores problemas, siendo que cada gasto era ingresado manualmente y de manera individual permitía a la equivocación seguido. Por ejemplo, estos tenían el número de factura guardado como un valor entero, pero si se le ingresaba un numero mal de la factura este ya formaba parte de una factura que no era. Continuando por este problema, si se ingresaba un digito de más o de menos en un precio unitario, este alteraba todos los valores de la semana y el mes, obligando al usuario a buscar entre todos los gastos y facturas cual fue mal ingresado, principalmente recurriendo a la memoria de cual puede haber sido.

B- = BARRA, C = COCINA, M- = MIXTO COCINA Y BARRA, SALON				
Fecha:	Comprado	Utilizado	En stock	REALIDAD
B-ALCOHOL				
BACARDI CARTA ORO 750ML	36	0	36	
BOSFORO 700ML	42	0	42	
CACHACHA VELHO BARREIRO 1LT	36	0	36	
CHAMPAGNE FED. DE ALVEAR EXTRA BRUT	72	0	72	
CHANDON BRUT	12	0	12	
DE LA COLONIA MERLOT ROSADO	48	0	48	
DE LA COLONIARIESLING VIognier	48	0	48	
DE LA COLONIA TEMPRANILLO MERLOT	72	0	72	
DE LA COLONIA TEMPRANILLO/TANNAT	150	0	150	
FERNET BRANCA 750CC	13	0	13	
GIN GORDON S (ING.)	12	0	12	
GRANADINA PRIMOR	2	0	2	
GREY GOOSE ORIGINAL 750ML	3	0	3	
J. WALKER E/NEGRA 1000CC	1	0	1	
J. WALKER E/ROJA 1000CC	1	0	1	
MALIBU (RON Y COCO) 700CC	3	0	3	
PDS CAB. SAUVIGNON	18	0	18	
PDS MALBEC	18	0	18	
PDS SAUVIGNON BLANC	18	0	18	
RON BACARDI BLANCO 750ML FINESA	1	0	1	
RON HAVANA BLANCO 3 AÑOS 750CC	60	0	60	
TEQUILA SAUZA GOLD 750CC	1	0	1	
TEQUILA SAUZA SILVER 750CC	1	0	1	
VODKA SKYY 750ML	24	12	12	
VODKA WYBOROWA 750CC	12	0	12	
WILLIAM LAWSONS C/EST 1LT	36	0	36	
B-BEBIDAS				
AGUA SALUS C/GAS 2.25L CHIHUA	4	0	4	
AGUA SALUS S/GAS 2.25L CHIHUA	1	0	1	
COCA COLA 2.25 PET	83	0	83	
COCA COLA LIGHT 2.25 PET	55	0	55	
JUGO NARANJA CITRIC 5L	310	0	310	
SALUS 6.25L	2	0	2	
SALUS 6.25L CHIHUA	27	0	27	
SCHW POMELO 1.5 PET	15	0	15	
SCHW TONICA 1.5 PET	25	0	25	
SPRITE 1.5 LT PET	4	0	4	
SPRITE 2.25 PET X6	30	0	30	
SPRITE S/A 1.5 PET	4	0	4	

Ilustración 4 - Prototipo: Planilla de stock

Los *stocks* semanales (Ilustración 4) requerían al analista tener todo ingresado para el sábado en la mañana sin olvidarse de ninguna factura ya que podría afectar al recuento de stock en el instante. Por ejemplo, con un producto del cual no hubiera stock hace semanas, este no aparecería en la planilla impresa para realizarlo llevando a realizar anotaciones a mano que pueden afectar a la comprensión del analista para ingresarla luego manualmente al sistema. Estos recuentos semanales tampoco permiten la información en tiempo real entre semana y requieren de varias horas realizándolo por semana.

Sector	Comprado toda la tempo.	Consumido toda la tempo.	Comprado (inicio-20/03)	Consumido (inicio-20/03)	Comprado entre 12/03-20/ Consumido entre 12/03-20	OBSERVACIONES
B-ALCOHOL	2,000.00	900.00	1,000.00	950.00	1,000.00	-50.00
B-BEBIDAS	1,500.00	500.00	1,000.00	450.00	500.00	50.00
B-CERVEZA					0.00	0.00
B-DESCARTABLES					0.00	0.00 Valor de consumo aproximado dada la dificultad de stockear s
B-HIELO					0.00	0.00
C-ALMACEN					0.00	0.00
C-AVES					0.00	0.00
C-CARNES					0.00	0.00
C-CONGELADOS					0.00	0.00
C-FRAMBRES					0.00	0.00
C-HELADOS					0.00	0.00
C-LACTEOS					0.00	0.00
C-PANADERIA					0.00	0.00
C-PESCADOS Y MARISCOS					0.00	0.00
C-VERDURAS					0.00	0.00
COMBUSTIBLES					0.00	0.00 No se stockea el combustible (Leña, gas, carbon), se considera
M-FRUTAS					0.00	0.00
M-HIERBAS					0.00	0.00
PROV. Temporales					0.00	0.00
SALON					0.00	0.00
Total Comprado/Consumido sin IVA:	3,500.00	1,400.00	2,000.00	1,400.00	1,500.00	0.00
IVA 10%	200	80	210	147	-10	-67
IVA 22%	100	40	90	63	10	-23
Monto Total:	\$3,800.00	\$1,520.00	\$2,300.00	\$1,610.00	\$1,500.00	-\$90.00
USD	88.37	35.35	53.49	37.44	34.88	-2.09
Personas toda la temporada:	10				5	
Iva incl.	\$8.84	\$3.53			\$6.98	-\$0.42
Comprado por persona					Comprado por persona	Consumido por persona
Personas 12/03-20/03:						
Iva incl.						
Comprado por persona						
Consumido por persona						

Ilustración 5 - Prototipo: Reporte por sector (Datos ficticios o vacíos)

A pesar de estos y varios otros defectos que tenía le permitía al analista rendir números a los altos cargos del hotel semana a semana trabajando 8 horas por semana (sin días libres) para realizarlo. Estos reportes (Ilustración 5) eran altamente informativos y permitían la comparación entre semanas en la mayoría de los sectores que podían variar continuamente, especialmente ante cambios de proveedores.

4.3. Lista de requerimientos

4.3.1. Historias de usuario

Las historias de usuario son una herramienta clave en la metodología ágil para el desarrollo de software, y su uso se basa en decisiones estratégicas que buscan mejorar la comprensión de las necesidades del usuario y facilitar la comunicación entre los distintos miembros del equipo de desarrollo. Estas constan de una descripción breve, informal y en lenguaje sencillo de lo que un usuario quiere lograr con el uso de un determinado producto de software.

El equipo optó por redactar los requerimientos en forma de historias de usuario siguiendo el enfoque BDD (Behavior Driven Development), lo cual facilita la comunicación entre los integrantes del proyecto, y pone al usuario en el centro de las decisiones de desarrollo. Se puede encontrar el detalle de BDD en el anexo [Behaviour Driven Development](#) [6]

Este formato permite describir, de manera clara, las funcionalidades del sistema desde la perspectiva de los usuarios finales, asegurando que cada historia refleja las necesidades y expectativas del cliente o usuario.

Cada historia de usuario se estructura mediante escenarios que explican, en términos de comportamiento, cómo debería funcionar el sistema en distintas situaciones, tanto en casos exitosos como en aquellos donde surgen fallos. Además, el hecho de incluir criterios de aceptación claros antes de comenzar el desarrollo permite al equipo minimizar los malentendidos, asegurando que todos comprendan qué se espera que el sistema haga.

El equipo utilizó este ciclo de vida porque facilita:

- Poner a los usuarios en el centro del proceso de desarrollo.
- Comprender el valor que cada funcionalidad aporta al producto.
- Evitar la duplicación de esfuerzos y posibles correcciones derivadas de errores de interpretación.
- Describir tanto escenarios de éxito como de fallo para garantizar que los comportamientos estén bien documentados.

Las historias de usuario se diseñan con base en los principios de BDD y cumplen con las características de I.N.V.E.S.T (mayor contexto en el anexo [I.N.V.E.S.T](#)): son independientes, negociables, valiosas, estimables, pequeñas y *testable*s. Cada historia está lo suficientemente acotada para ser completada dentro de una iteración de desarrollo, lo que garantiza un ciclo de entrega ágil y continuo.

4.3.2. Requerimientos principales funcionales

Historia de usuario CC-6: Automatización de la captura de datos de facturas

Como usuario de la app web,

Quiero que el sistema capture automáticamente los datos de las facturas electrónicas,

Para que el ingreso de datos de compras sea rápido, automatizado y sin errores manuales.

Criterios de aceptación:

- El sistema debe realizar el ingreso de facturas desde el proveedor a la base de datos de manera automática.
- El sistema debe realizar el ingreso de facturas únicamente de los proveedores ingresados.
- El sistema debe ejecutar este proceso periódicamente en segundo plano.

Historia de usuario CC-11: Gestión de proveedores

Como usuario de la *app web*,

Quiero gestionar la carga, actualización, listado y eliminación de proveedores,

Para mantener un control sobre las empresas que nos venden los insumos necesarios para operar.

Criterios de aceptación:

- El sistema debe permitir el ingreso de nuevos proveedores con su RUT y nombre.
- El sistema debe permitir la modificación de proveedores ya ingresados.
- El sistema debe permitir la visualización de los proveedores ya ingresados.
- El sistema debe permitir la habilitación o inhabilitación de un proveedor.
- El sistema debe informar ante cualquier fallo en el ingreso, modificación o consulta de valores de los proveedores.

Historia de usuario CC-15: Gestión de productos

Como usuario de la *app web*,

Quiero gestionar la carga, actualización, listado y eliminación de productos,

Para mantener un inventario preciso y actualizado que refleje las necesidades del negocio.

Criterios de aceptación:

- El sistema debe permitir el ingreso de nuevos productos con su nombre.
- El sistema debe permitir la modificación de productos ya ingresados.
- El sistema debe permitir la visualización de los productos ya ingresados.
- El sistema debe permitir la habilitación o inhabilitación de un producto.
- El sistema debe informar ante cualquier fallo en el ingreso, modificación o consulta de valores de los productos.

Historia de usuario CC-171: Gestión de usuarios

Como usuario de la *app web*,

Quiero gestionar la creación, actualización, listado y eliminación de usuarios,

Para asegurarme de que solo personas autorizadas tengan acceso al sistema y sus funcionalidades.

Criterios de aceptación:

- El sistema debe permitir el ingreso de nuevos usuarios con su nombre, apellido, usuario, email y contraseña.
- El sistema debe permitir la modificación de usuarios ya ingresados.
- El sistema debe permitir la visualización de los usuarios ya ingresados.
- El sistema debe permitir el eliminado físico de un usuario.
- El sistema debe informar ante cualquier fallo en el ingreso, modificación o consulta de valores de los usuarios.

Historia de usuario CC-117: Login

Como usuario de la *app web*,

Quiero poder iniciar y cerrar sesión utilizando mi nombre de usuario y contraseña,

Para asegurar el acceso a mi cuenta personal y proteger mi información privada.

Criterios de aceptación:

- El sistema debe permitir el ingreso de un usuario a partir de un nombre de usuario y contraseña.
- El sistema debe retornar un token JWT con el cual se realizarán todas las consultas por un determinado periodo.
- El sistema deberá rechazar el ingreso de un usuario con sus credenciales incorrectas sin informar cual campo fue el incorrecto.
- El sistema debe informar en caso positivo o negativo de la acción.

Historia de usuario CC-159: Historial de compras

Como usuario de la *app web*,

Quiero poder visualizar el historial de compras,

Para tener control sobre las compras que se le hacen a los proveedores.

Criterios de aceptación:

- El sistema debe permitir la visualización de todas las facturas ingresadas en el sistema.
- El sistema debe permitir la eliminación de una factura.
- El sistema debe permitir el filtrado por fecha desde y hasta de la factura.

Historia de usuario CC-17: Registro de ingreso de mercadería

Como usuario de la *app* móvil,

Quiero agregar un nuevo movimiento de mercadería en el sistema,

Para que se mantenga el control del stock y prevenir desabastecimientos.

Criterios de aceptación:

- El sistema debe permitir registrar el ingreso de mercadería de un producto seleccionado.
- El sistema debe permitir indicar la cantidad de su ingreso.
- El sistema debe avisar si el ingreso fue logrado o fallido.

Historia de usuario CC-18: Registro de egreso de mercadería

Como usuario de la *app* móvil,

Quiero agregar un nuevo egreso de mercadería en el sistema,

Para que se mantenga el control del stock.

Criterios de aceptación:

- El sistema debe permitir registrar el egreso de mercadería de un producto seleccionado.
- El sistema debe permitir indicar la cantidad de su egreso.
- El sistema debe avisar si el egreso fue logrado o fallido.

4.3.3. Requerimientos no funcionales

RNF-1 – Seguridad

El sistema debe garantizar la seguridad de los datos sensibles mediante el uso de cifrado de datos tanto en tránsito como en reposo (protocolo HTTPS, tokens JWT para las sesiones de usuario y cifrado de datos sensibles).

RNF-2 – Disponibilidad

El sistema debe tener una disponibilidad mínima del 99% del tiempo en que debe ser utilizado durante la temporada alta, asegurando que los usuarios puedan acceder al sistema en cada momento, excepto durante los períodos destinados al mantenimiento del producto.

RNF-3 – Escalabilidad

El sistema debe ser capaz de escalar horizontalmente para soportar un aumento en la carga de trabajo durante la temporada alta (en caso de ser necesario), para no afectar el rendimiento del usuario.

RNF-4 – Rendimiento

El sistema debe permitir que el ingreso o modificación de datos, transcurra en un tiempo promedio que no supere los 2.5 segundos promedio, garantizando una experiencia de usuario rápida y sin demoras. Las consultas de datos deben realizarse en un tiempo menor a los 3 segundos promedio.

RNF-5 – Usabilidad

Las interfaces de usuario (tanto de la aplicación del contenedor, como el panel admin web), debe ser intuitiva, siguiendo los principios de diseño centrado en el usuario (heurísticas de Nielsen). La interfaz debe proporcionar retroalimentación a los usuarios sobre el estado de sus acciones, incluyendo mensajes de error claros y confirmaciones (siempre y cuando sea necesario). El sistema debe estar accesible para usuarios con diferentes niveles de habilidad tecnológica.

La primera temporada del hotel, en la cual el sistema estará implementado (temporada que abarca el período de tiempo entre los meses de noviembre y fines de abril), será también una prueba piloto para el sistema. El equipo no considera que el análisis para lograr la mejor usabilidad posible culmina al momento de hacer el *release* a producción, sino que este aspecto seguirá evolucionando a medida que pasa el tiempo y el sistema se va utilizando. Por eso se puede considerar a la primera temporada de utilización de ChefCheck-Manager, como una temporada piloto.

RNF-6 – Compatibilidad

El panel web admin debe ser compatible con los navegadores web más populares, teniendo como referencia a Google Chrome (por el acceso desde Windows, Android e iOS). Además, las diferentes pantallas de la web deben de poder adaptarse a distintas dimensiones de pantalla, como dispositivos móviles, tablets y computadoras.

Respecto a la aplicación del contenedor, la misma debe ser únicamente compatible con el sistema operativo Android (en un principio), y desarrollada para ser utilizada en una tablet, en su modalidad horizontal (*landscape*). Debe desarrollarse por medio de un *framework* multiplataforma, debido a que si en un futuro, es necesario hacer un *release* de una versión iOS, se puede hacer con un costo muy bajo.

RNF-7 – Extensibilidad

El sistema debe permitir la extensibilidad, para permitir la integración de nuevos módulos y funcionalidades en el futuro, como, por ejemplo, la incorporación de nuevas tecnologías de *machine learning* para mejorar las predicciones de consumo y análisis de datos.

RNF-8 – Mantenibilidad

El sistema debe realizarse bajo una arquitectura modular, para facilitar su mantenimiento y actualización en el futuro. El código debe estar documentado adecuadamente y seguir estándares de codificación reconocidos para asegurar la facilidad de comprensión y modificación. Las actualizaciones (puesta en producción), deben poder realizarse sin causar interrupciones significativas en el servicio. Como la operativa del hotel no dura las 24 horas, existen varios momentos para realizar estas actualizaciones.

RNF-9 – Recuperación ante fallos

La infraestructura de *hosting* debe contar con mecanismos de recuperación ante fallos que permitan restaurar su funcionamiento en caso de incidentes críticos. Debe incluir copias de seguridad regulares de los datos y procedimientos claros para la recuperación rápida y efectiva.

RNF-10 – Integración

El sistema debe integrarse sin problemas con los servicios externos de la empresa Factura Lista para la descarga e ingreso automático de datos de facturación electrónica.

RNF-11 – Monitorización y registro

El sistema debe registrar información sobre los errores ocurridos (error, tipo de error, fecha, usuario, etc.), y deben ser almacenados de manera segura en la base de datos. Esto facilita al equipo de desarrollo, la determinación del tipo de error que ocurrió, lo cual agiliza el soporte para solucionarlos.

4.3.4. Entregables

Los entregables correspondientes a componentes del sistema se detallan en la sección [Arquitectura a alto nivel](#):

- Base de datos de negocio.
- Base de datos para módulo de *machine learning*.

- *Script de machine learning.*
- *Web API* (incluyendo una integración con la API de la empresa FacturaLista).
- *Frontend web.*
- *Frontend mobile.*
- Despliegue en Azure para las bases de datos, *script de machine learning*, *web API* y *frontend web*.
- Despliegue en Google Play Store para aplicación *mobile*.
- Documentación técnica del sistema (requerimientos funcionales/ no funcionales, decisiones técnicas, diagramas, descripciones de los componentes de la arquitectura y su comunicación, etc).

4.4. Validación de problema y solución

En el desarrollo de software, es fundamental entender la diferencia entre validar y verificar tanto los requerimientos como las soluciones propuestas.

- **Validar el problema:** Este proceso implica asegurarse de que tanto el problema como la solución identificados realmente reflejan las necesidades del cliente o usuario. A través de entrevistas y sesiones de retroalimentación, se busca confirmar que el problema planteado es relevante y que los requerimientos recopilados son correctos y adecuados para abordar dicha problemática. Esta validación es esencial para garantizar que el enfoque del proyecto esté alineado con las expectativas y necesidades del cliente, lo que establece una base sólida para el desarrollo de una solución efectiva.

- **Verificar la solución:** Una vez que se ha desarrollado una solución o un prototipo, la verificación se enfoca en evaluar si está realmente cumple con los requerimientos establecidos y resuelve el problema identificado. En esta fase, se presenta el producto al cliente para que valide que la solución corresponde a sus expectativas y necesidades originales.

Ambos procesos son cruciales para asegurar que el desarrollo del software no solo aborda el problema real, sino que también ofrece una solución efectiva y alineada con las expectativas del cliente, minimizando así el riesgo de malentendidos durante el proyecto.

La validación del problema se realizó en primera instancia en la visita del equipo al hotel. Posteriormente se realizaron reuniones y se mantuvo un contacto directo con el designado del hotel para que nos comunicara cualquier idea nueva que se presentara. De esta también fueron parte el nuevo encargado de alimentos y bebidas, un consultor externo y el dueño en distintas instancias.

La verificación del prototipo fue realizada en los veranos de 2020, 2021 y 2022, recibiendo un *feedback* positivo de lo que visualizaban los gerentes que eran las planillas de stock y los reportes por sector. Toda crítica era bienvenida y en lo posible solucionada en la implementación para su validación y verificación nuevamente en las siguientes entregas hasta la finalización del verano en cuestión. La verificación del nuevo sistema se estaría realizando en [próximas](#) instancias.

5. Arquitectura de la solución

5.1. Introducción a la arquitectura

Este capítulo se centra en la arquitectura del sistema *ChefCheck-Manager*, diseñada para mejorar la gestión de costos e inventarios en el hotel Tio Tom *All Inclusive*. La arquitectura es un elemento clave que facilita la eficiencia operativa del hotel al ofrecer un control detallado del inventario y proporcionar análisis de datos que apoyan la toma de decisiones.

La relevancia de la arquitectura del sistema se hace notar en su capacidad para cumplir con varios atributos de calidad esenciales para el éxito del proyecto. Estos incluyen la eficiencia en el manejo de datos, la escalabilidad para ajustarse a diferentes volúmenes de trabajo, la seguridad en el acceso a información sensible, la mantenibilidad del sistema a lo largo del tiempo, y la integración con otros sistemas, etc.

La solución se estructura en varios componentes principales: *frontend*, *backend*, y módulos de *machine learning* e integración con servicios externos. Cada componente tiene un rol específico y se comunica de manera eficiente para asegurar una experiencia de usuario óptima. En este capítulo, se describirán estos componentes y las decisiones de diseño que han guiado su implementación.

El sistema adopta una arquitectura de microservicios, donde cada servicio gestiona una funcionalidad particular, como la gestión de usuarios, inventarios, y generación de reportes. Este enfoque facilita la escalabilidad y el mantenimiento del sistema, además de permitir actualizaciones sin mayores inconvenientes. El despliegue en la nube, mediante Microsoft Azure, ofrece la flexibilidad y seguridad necesarias para mantener los datos siempre accesibles y protegidos.

Asimismo, *ChefCheck-Manager* incorpora prácticas de desarrollo como pruebas automáticas e integración continua, ayudando a garantizar la calidad y estabilidad del sistema en el tiempo. La flexibilidad de esta arquitectura permite futuras mejoras o incorporaciones de nuevas funcionalidades, adaptándose a las necesidades evolutivas del hotel. Este enfoque asegura que *ChefCheck-Manager* no solo cumple con los requisitos actuales del cliente, sino que también está preparado para posibles expansiones futuras.

5.2. Arquitectura basada en microservicios

En el campo de la tecnología de la información, la arquitectura de microservicios se ha convertido en una opción popular por su capacidad para adaptarse y escalar en diferentes contextos. En el caso de *ChefCheck-Manager*, los microservicios son fundamentales.

Los microservicios consisten en dividir una aplicación en partes más pequeñas y autónomas, donde cada una cumple una función específica. A diferencia del enfoque tradicional y monolítico, donde todo está interconectado y es difícil de modificar sin afectar otras partes, los microservicios permiten que cada componente funcione de manera independiente. Esto facilita el mantenimiento y la mejora continua del sistema.

En *ChefCheck-Manager*, esta estructura basada en microservicios no solo simplifica la implementación y el mantenimiento, sino que también clarifica las responsabilidades de cada parte del sistema. Cada microservicio tiene un rol definido y opera por separado, pero todos colaboran para cumplir con los objetivos del sistema. Esta independencia es clave porque permite a los desarrolladores actualizar, desplegar o escalar cada servicio sin interferir con los demás, algo esencial para mantener la flexibilidad y la capacidad de respuesta del sistema.

La elección de una arquitectura de microservicios para el sistema no es por moda, sino que es una decisión pensada para asegurar que el sistema pueda responder a las necesidades del proyecto. Esto incluye la capacidad de escalar según la demanda, facilitar el mantenimiento y la actualización, y la posibilidad de integrar nuevas tecnologías o módulos sin interrupciones importantes. Esta decisión permitió que el sistema se mantenga eficiente y adaptable en el entorno dinámico de un hotel *all inclusive*.

5.2.1. Justificación de la utilización de una arquitectura basada en microservicios

La elección de este tipo de arquitectura se realizó luego de un análisis, el cual busca balancear las necesidades inmediatas del sistema con las proyecciones de crecimiento futuro. En el caso de *ChefCheck-Manager*, la decisión de adoptar una arquitectura basada en microservicios fue producto de un análisis estratégico. A continuación, se detallan las razones clave para esta elección, demostrando cómo cada una, de cierta manera, se ve relacionada con los atributos de calidad y la visión a largo plazo del sistema:

Escalabilidad

La arquitectura de microservicios permite escalar cada componente de manera independiente según la demanda. Por ejemplo, cada unidad que conforma nuestro sistema puede ser escalado sin necesidad de incrementar recursos en otras áreas del sistema, lo cual optimiza el uso de recursos y reduce costos operativos.

Tolerancia a fallos

Cada microservicio funciona de manera autónoma, lo que significa que un fallo en un servicio no necesariamente afecta a los demás. Esto mejora la resiliencia del sistema y permite una recuperación más rápida de fallos, asegurando la continuidad del negocio y la disponibilidad del sistema para los usuarios.

Flexibilidad tecnológica

La independencia de cada microservicio permite elegir las tecnologías más adecuadas para cada función específica. *ChefCheck-Manager*, por ejemplo, utiliza .NET Core para su API y Python para el análisis de datos y *machine learning*. Esta flexibilidad asegura que cada componente pueda ser optimizado y actualizado de manera independiente, aprovechando las mejores herramientas disponibles.

Cada componente del sistema (contemplando también el *frontend*), son realizados en tecnologías diferentes.

Desarrollo independiente y despliegue rápido

La estructura de microservicios permite que diferentes equipos puedan trabajar en paralelo en distintos servicios, acelerando el desarrollo y permitiendo una entrega continua de nuevas funcionalidades. Esto es vital para mantener el sistema actualizado y responder rápidamente a las necesidades que puedan surgir.

Seguridad y cumplimiento de las normas

Al tener servicios específicos encargados de realizar funciones sensibles, como la gestión de datos de usuarios y la facturación, es posible implementar medidas de seguridad y políticas de cumplimiento. Esto asegura que el sistema cumpla con las regulaciones vigentes y proteja los datos sensibles de los usuarios.

Interoperabilidad

La clara separación de responsabilidades entre los microservicios facilita la integración con sistemas externos y la expansión de funcionalidades. El sistema puede integrarse fácilmente con proveedores de servicios externos, sistemas de facturación electrónica y módulos adicionales, asegurando la adaptación a nuevos requerimientos del mercado.

La arquitectura seleccionada por el equipo no solo contempla las necesidades actuales del sistema, sino que también prepara el sistema para futuras necesidades. Este enfoque garantiza que el sistema pueda crecer y adaptarse, ofreciendo un servicio de alta calidad y eficiencia a largo plazo.

5.2.2. Arquitectura a alto nivel

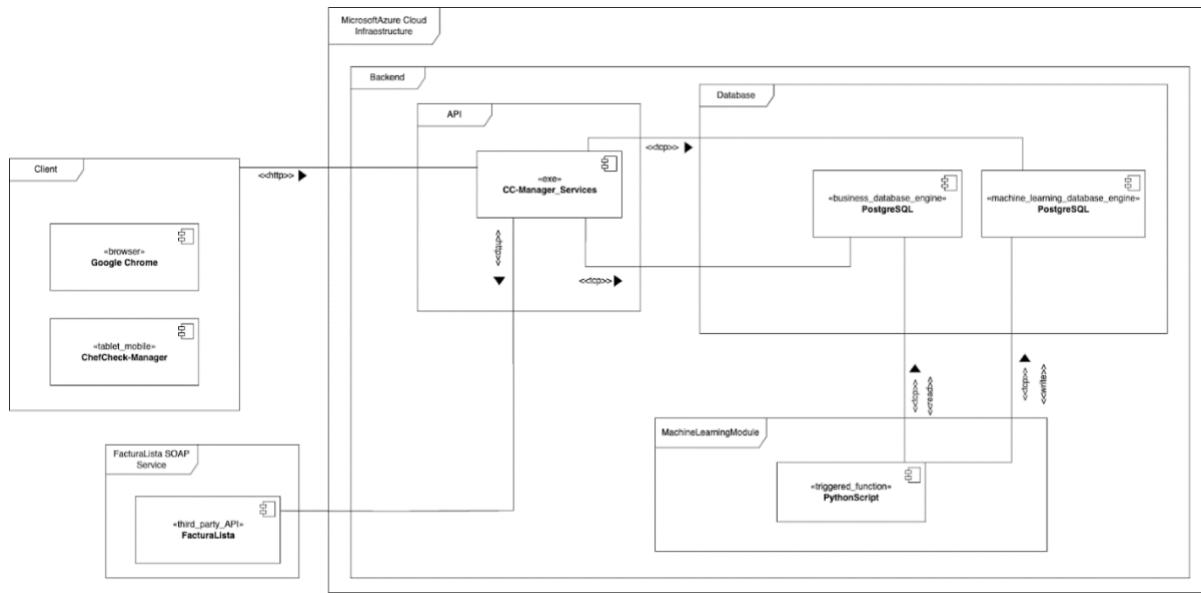


Ilustración 6 - Arquitectura a alto nivel

Este capítulo describe la arquitectura de alto nivel (Ilustración 6) de *ChefCheck-Manager*, destacando su diseño modular y orientado al usuario. La estructura arquitectónica se ha diseñado para garantizar escalabilidad, mantenibilidad y eficiencia en la gestión de inventarios y costos. A continuación, se desglosan los componentes clave del sistema y se explica su función dentro del ecosistema general.

Los diferentes módulos existentes son:

Frontend

El *frontend* de *ChefCheck-Manager* incluye interfaces de usuario tanto *web* (accesible a través de navegadores como Google Chrome) como móvil. Estas interfaces son cruciales para proporcionar una experiencia de usuario intuitiva y accesible, facilitando la interacción con el sistema para gestionar inventarios, consultar reportes y recibir alertas.

Backend

Esta capa maneja la lógica de negocio del sistema y coordina la interacción entre el *frontend* y los demás componentes. Los servicios centrales procesan las solicitudes de los usuarios, gestionan la lógica de inventarios y compras, y orquestan las comunicaciones con las bases de datos y otros sistemas.

Módulo de *machine learning*

Un componente especializado (script de Python) que ejecuta algoritmos de *machine learning* para analizar datos históricos de consumo y realizar predicciones sobre demandas futuras. Este módulo ayuda a optimizar el inventario y la planificación de compras, asegurando que el hotel mantenga un nivel adecuado de stock y minimice las pérdidas. Este script ejecuta una función automática de Azure cada cierto período de tiempo, en el cual primero entrena al algoritmo con los datos, y luego almacena en la base de datos, un reporte con el informe.

Este módulo fue descartado para la versión inicial del sistema que será entregado al cliente. El mismo fue definido únicamente a alto nivel, y no en profundidad como para poder comenzar su desarrollo a corto plazo.

La razón de su descarte se debe a que el cliente quiere que el equipo centre los esfuerzos en aquellos requerimientos clave, como, por ejemplo, el data entry automático de las facturas, así como la gestión en tiempo real del inventario. Además, para que este módulo genere resultados de valor, se debe tener datos de al menos una o dos temporadas, por lo que no es viable a fecha de hoy.

Por eso, este requerimiento, no forma parte del ciclo de vida del proyecto de la facultad, sino que el mismo comenzará a implementarse, luego de la finalización de la próxima temporada del hotel (la cual comienza en noviembre de 2024, y finaliza en abril de 2025).

Bases de datos

Se cuenta con dos bases de datos:

La base de datos de negocio (PostgreSQL), se encuentra alojada en la infraestructura de la nube de Microsoft Azure, donde se almacenan los datos relacionados con productos, proveedores, transacciones y usuarios, etc. Esta base de datos está optimizada para manejar grandes volúmenes de datos y proporcionar acceso rápido y seguro a la información crítica.

El equipo cuenta con una segunda base de datos PostgreSQL (la cual se ejecutará en una instancia diferente a la base anterior). La misma será utilizada como repositorio de los datos consecuencia de los algoritmos de *machine learning*. Esta base de datos almacenará el output producido por las diversas predicciones realizadas por los modelos utilizados.

Servicio de Integración con FacturaLista

Este servicio de terceros, el cual es consumido por la *API* de *ChefCheck-Manager*, se encarga de la integración con el sistema de facturación electrónica de la DGI, automatizando el ingreso de datos de facturas al sistema. Esto reduce la necesidad de entrada manual de datos y mejora la precisión de la información de costos y compras.

5.2.3. Importancia de la separación en tres componentes (*frontend*, *backend* y servicio de FacturaLista)

Como anteriormente el equipo mencionó, la arquitectura de *ChefCheck-Manager* está diseñada con tres componentes principales: *frontend*, *backend* y el servicio de FacturaLista. Esta estructura modular facilita la gestión eficiente del sistema, permitiendo una clara división de responsabilidades y funciones. Cada componente está gestionado de manera independiente, asegurando una implementación y mantenimiento eficientes.

5.2.3.1. *Frontend*

Responsabilidad principal

El *frontend* del sistema, es la interfaz de usuario que interactúa directamente con los usuarios del sistema. Es el punto de contacto donde los usuarios realizan operaciones como la gestión de inventarios, consulta de reportes y recepción de alertas.

Algunas de sus características clave

Fue diseñado para ser intuitivo y amigable (debido a que sus usuarios no son técnicos), el *frontend* ofrece una experiencia de usuario optimizada. Es accesible desde múltiples dispositivos, incluyendo navegadores *web* (como Google Chrome) y dispositivos móviles (mediante la aplicación *ChefCheck-Manager*). La interfaz está adaptada para proporcionar una interacción fluida y eficiente. Tanto la aplicación *web* como móvil, son dos productos diferentes, con funcionalidades diferentes, y para usuarios diferentes.

Comunicación

Ambos componentes *frontend*, se comunican con el *backend* mediante solicitudes HTTP, lo que permite enviar y recibir datos de manera segura y eficiente, facilitando una respuesta rápida a las solicitudes de los usuarios.

5.2.3.2. *Backend*

Responsabilidad principal

El *backend* es el componente central que maneja la lógica de negocio del sistema. Gestiona funciones críticas como la autenticación de usuarios, el almacenamiento y procesamiento de datos, y las integraciones con otros servicios.

Algunas de sus características clave

El *backend* implementa una *API REST* que facilita la comunicación con el *frontend* y otros sistemas. Este componente es responsable de asegurar la integridad y seguridad de los datos, así como en un futuro, soportar los procesos de análisis y *machine learning* que optimizan la gestión de inventarios y previsión de necesidades.

Comunicación

Interactúa con el *frontend* para gestionar las solicitudes de los usuarios y con servicios externos como FacturaLista para la automatización de la facturación electrónica. También se encarga de coordinar el flujo de datos entre diferentes módulos del sistema.

Con el servicio de FacturaLista, la comunicación es también realizada por medio del protocolo HTTP.

La *API*, por medio del protocolo TCP/IP, se comunica con las bases de datos PostgreSQL.

5.2.3.3. Servicio de FacturaLista

Responsabilidad principal

FacturaLista Service es un módulo de terceros (perteneciente a la empresa con mismo nombre), que *ChefCheck-Manager* integra para gestionar la facturación electrónica de manera automática. Este componente es crucial para manejar las transacciones y registros financieros con precisión y eficiencia (para que haya la menor intervención posible del cliente).

Algunas de sus características clave

Este servicio se encarga de la integración con el sistema de facturación electrónica de la DGI, permitiendo el ingreso automático de datos de facturas y reduciendo la necesidad de intervención manual. Esto ayuda a mejorar la precisión de los registros y a mantener un control riguroso de los costos y compras. Únicamente se va a trabajar con las facturas emitidas hacia el hotel, (no las emitidas por el hotel). En principio serán únicamente aquellas vinculadas a gastos en alimentos y bebidas.

Comunicación

El *backend* del sistema se comunica directamente con el servicio *SOAP* de FacturaLista, proporcionando los datos de facturación necesarios para la contabilidad y gestión financiera del sistema.

Estos tres componentes trabajan en conjunto para hacer de *ChefCheck-Manager*, un sistema robusto y eficiente.

5.2.4. Descripción de la comunicación entre componentes

La comunicación entre los componentes del sistema es fundamental para garantizar un funcionamiento eficiente y lograr la cohesión del sistema. La *API REST* es el principal medio de comunicación entre el *frontend* y todo lo que compone el servidor, permitiendo que los datos se intercambien por medio del formato JSON. Este formato es esencial para la integración entre los diferentes módulos, facilitando operaciones como la consulta de inventarios, la actualización de datos, la generación de reportes, etc. Las solicitudes HTTP (*GET, POST, PUT, DELETE*) son utilizadas para realizar estas operaciones, asegurando una interacción fluida y en tiempo real entre la interfaz del usuario y la lógica de negocio.

La comunicación desde el *frontend* hacia la *API* del *backend* se realiza a través del protocolo HTTP, que es clave para enviar datos desde el cliente y recibir las respuestas procesadas por el *backend*. Este flujo de información es crucial para mantener una experiencia de usuario eficiente y accesible, permitiendo que las acciones iniciadas en el *frontend* se reflejen correctamente en el sistema. Por otro lado, el *backend* se comunica con ambas bases de datos PostgreSQL utilizando el protocolo TCP/IP. Para obtener todo lo relacionado a la operativa del negocio, la *API* se comunica con la base de datos de negocio. Por otro lado, para obtener todo lo relacionado a informes de predicciones de datos, la *API* se comunicará con la base de datos dedicada a almacenar los datos producto de los algoritmos de *machine learning*. Esta conexión asegura una transmisión de datos confiable y segura, manejando tanto la inserción como la consulta de datos de manera eficiente, lo que es fundamental para mantener la integridad y consistencia de la información almacenada.

Por otro lado, el módulo de *machine learning* (contenido en un script Python), se comunicará únicamente con ambas bases de datos por medio del protocolo TCP/IP. Con relación a la base de datos de negocio, el script Python recuperará dichos datos de negocio (los cuales han sido acumulados a lo largo del tiempo). Con los datos históricos, el script procederá a realizar las operaciones de extracción, transformación y carga de los datos. Una vez culminados los procesos anteriores, los diferentes modelos de *machine learning* se podrán ejecutar. El output de la ejecución de modelos de *machine learning*, son datos estructurados/estandarizados, y, por ende, serán almacenados en la segunda base de datos PostgreSQL dedicada exclusivamente a este fin.

ChefCheck-Manager utiliza el protocolo HTTPS para intercambiar solicitudes con la *API* de terceros proporcionada por FacturaLista.

La arquitectura de *ChefCheck-Manager* favorece la comunicación asíncrona, lo que significa que las solicitudes pueden ser procesadas de manera independiente y simultánea. Este enfoque mejora la capacidad de respuesta del sistema, permitiendo que múltiples operaciones se lleven a cabo sin interrupciones, especialmente durante momentos de alta demanda. Además, toda la comunicación, tanto interna como externa, se tiene la intención de que se realice por medio de HTTPS, asegurando que los datos transmitidos estén cifrados y protegidos contra accesos no autorizados. Este nivel de

seguridad es crucial para proteger la información sensible y garantizar la integridad de las transacciones y datos manejados por el sistema.

A continuación, se mostrarán diagramas de secuencia de aquellos casos de uso claves del sistema, para poder visualizar cómo es la interacción dentro de los componentes de la arquitectura. La idea de los diagramas es ilustrar el caso de uso no a muy bajo nivel, sino a un cierto nivel que se pueda comprender como es la interacción entre los componentes, pero sin profundizar dentro de cada uno.

5.2.4.1. Agregación de productos al sistema

Es el caso de uso que da sentido a todo en el sistema, ya que los productos (en principio alimentos y bebidas) son el propósito del sistema. Es la entidad de negocio más importante.

A continuación, se puede observar el detalle de un diagrama de secuencia (Ilustración 7) para el caso de uso de agregación de nuevos productos [4] [5].

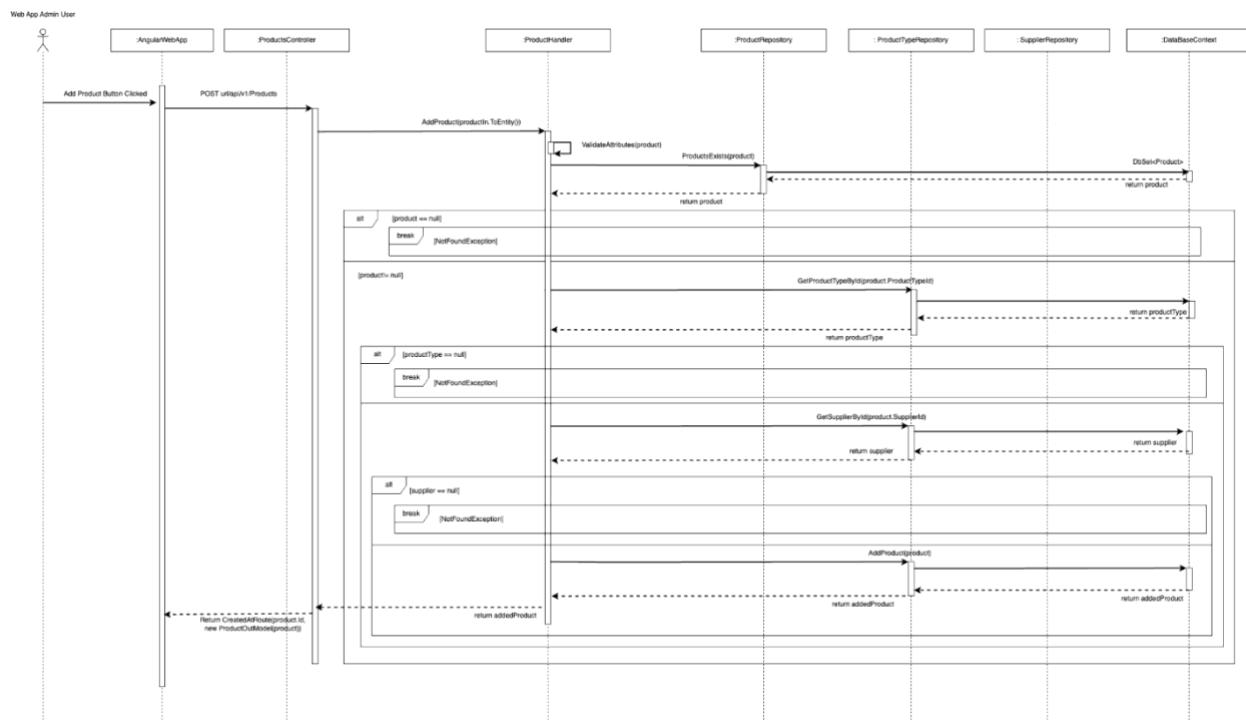


Ilustración 7 - Diagrama de secuencia de agregación de nuevos productos

5.2.4.2. Egreso de mercadería del contenedor

Este caso de uso (Ilustración 8), junto con el anteriormente descrito (Ilustración 7), es quizás uno de los más importantes del sistema, debido a que, si un producto ingresado no es extraído del contenedor de almacenamiento de productos, no es marcado como consumido en el sistema. Entonces, para que los reportes de *machine learning* puedan predecir los consumos a lo largo del

tiempo, es imprescindible que cuando un producto se extraiga del contenedor, el mismo sea marcado como egresado en el sistema.

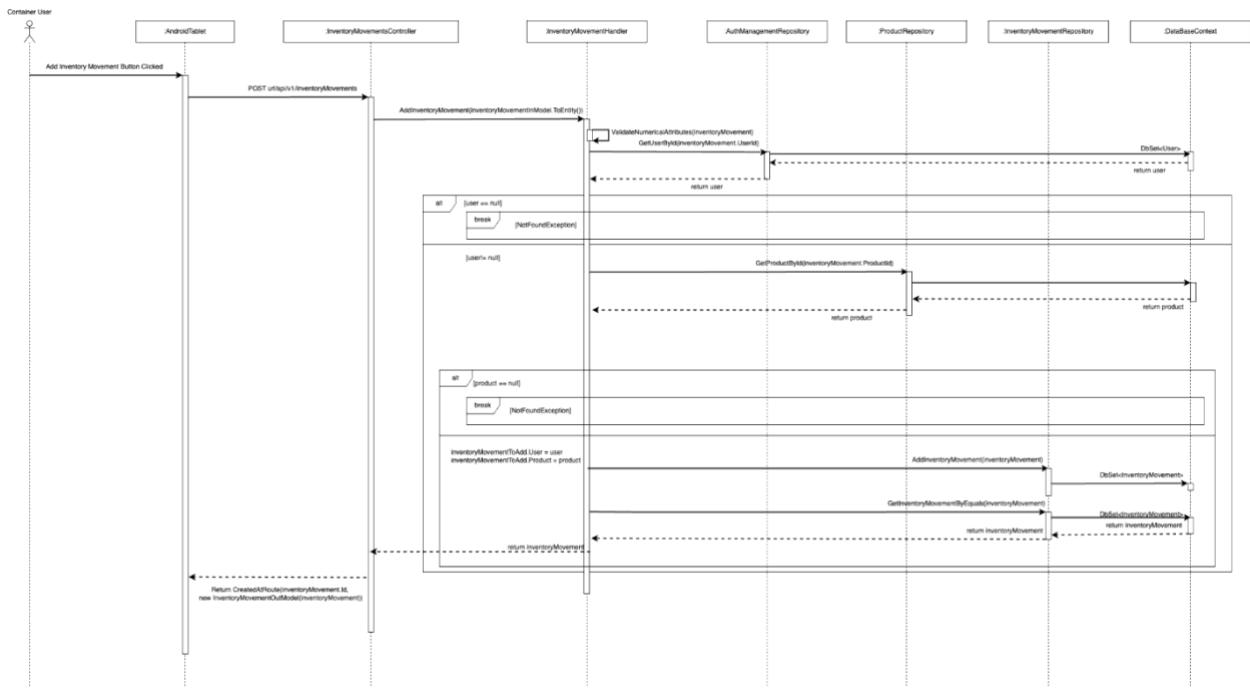


Ilustración 8 - Diagrama de secuencia de egreso del contenedor

No se debe confundir este caso de uso con el anterior. Este caso de uso registra los movimientos de inventario para conocer el uso que se le da a dicho producto. El caso de uso anterior lo agrega al sistema (son acciones diferentes).

En este escenario, se ve involucrado un usuario que utiliza el cliente móvil de *ChefCheckManager*, el cual, luego de extraer (por ejemplo) veinte tomates del contenedor, deja el registro del movimiento en el sistema. El cliente se comunica con la *API*, realizando las validaciones correspondientes de los datos de entrada (por ejemplo, que las cantidades no sean negativas), para luego validar el movimiento, y enviarlo a la base de datos PostgreSQL.

Para que la operación anterior pueda suceder, el usuario debe enviar desde el cliente hacia la *API*, un token, el cual será verificado en la misma *API*, para que, en el caso de su validez, permita completar la petición. Dicho token tiene datos importantes para validar al usuario, como, por ejemplo: timeout y nombre de usuario que realizó la petición.

5.2.4.3. Obtención de facturas correspondiente a gastos del Hotel

Este caso de uso describe el proceso de obtención de facturas desde el servicio *SOAP* de FacturaLista.

La empresa FacturaLista es uno de los tantos proveedores de facturación electrónica que existen en el Uruguay. Esta empresa ya trabajaba con el hotel, por ende, el cliente nos solicitó que trabajemos con ellos.

FacturaLista actualmente dispone de un servicio *SOAP* (la *API REST* no es estable de momento).

El primer paso de la comunicación es la autenticación con el servicio (Ilustración 9). Se llama a un *endpoint* y se le envía un body, con datos como usuario y contraseña. Dentro del body de la respuesta obtenida ante la llamada, se encuentra el campo *tokensessiom*, que -como indica su nombre- será el que contenga el token necesario.

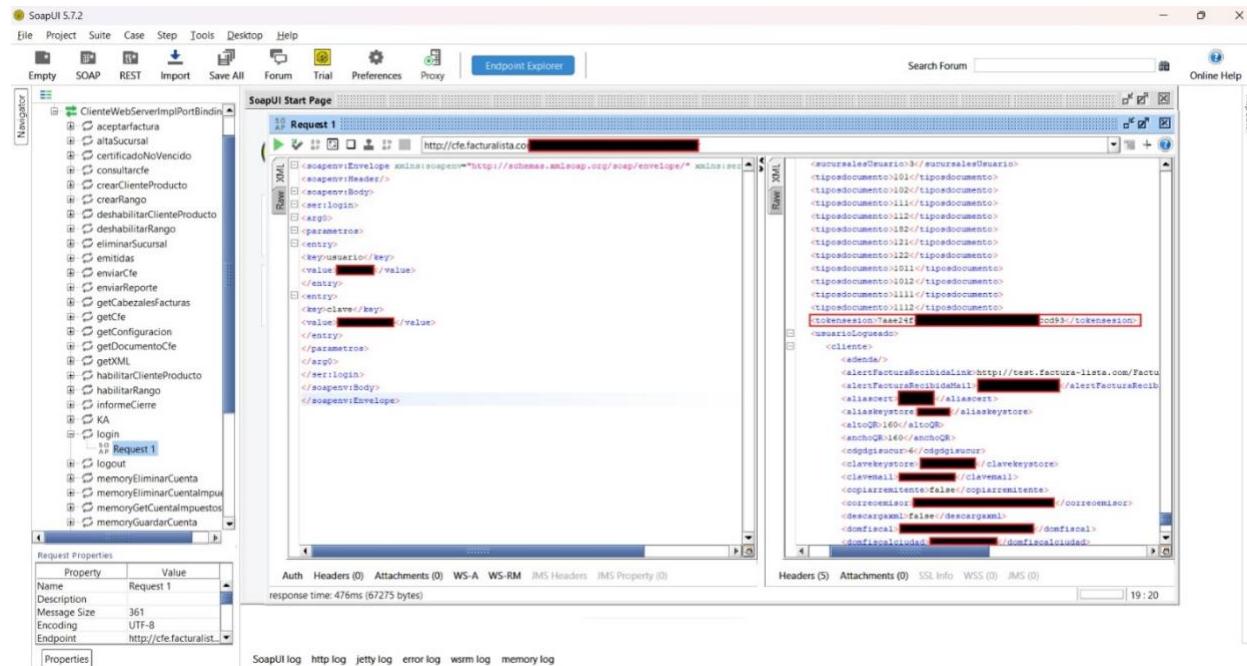


Ilustración 9 - Autenticación con FacturaLista (datos sensibles censurados)

Una vez conseguido ese token, el equipo puede proceder con la comunicación con el servicio y consultar las facturas. Para eso, FacturaLista dispone de otro *endpoint* que recibe tres parámetros: fecha desde, fecha hasta, y el token para la autenticación (Ilustración 10). La respuesta obtenida es el listado de facturas emitidas al hotel, dentro del rango de fechas especificado.

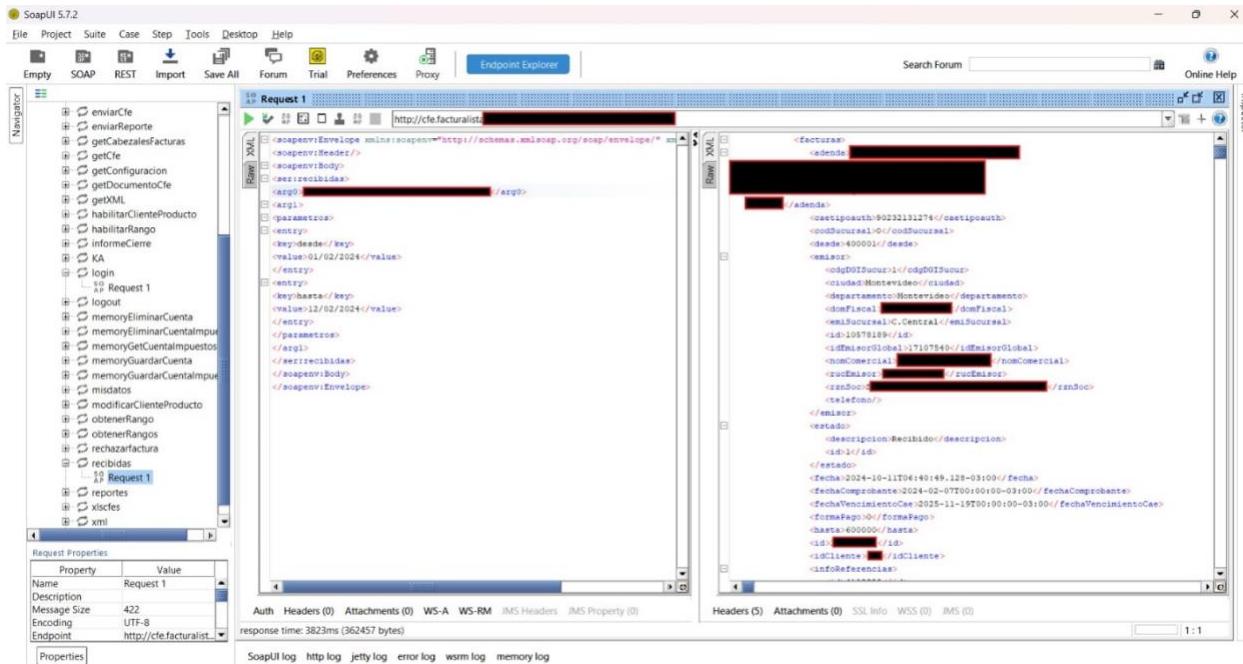


Ilustración 10 - *Output* al consultar las facturas en un rango de fechas (datos sensibles censurados)

El diagrama (Ilustración 11) sólo muestra la obtención de las facturas del hotel mediante el proveedor de facturación electrónica, junto a la agregación (a alto nivel) de las mismas en la base de datos del sistema del hotel. No se detalla a bajo nivel la autenticación con FacturaLista, ni tampoco la creación y almacenamiento de los nuevos productos que deriven de esas facturas, ya que es la misma lógica que se aplica para el ingreso manual de facturas. Solo se busca demostrar, para un mayor entendimiento, cómo funciona la integración con el proveedor para obtener las facturas.

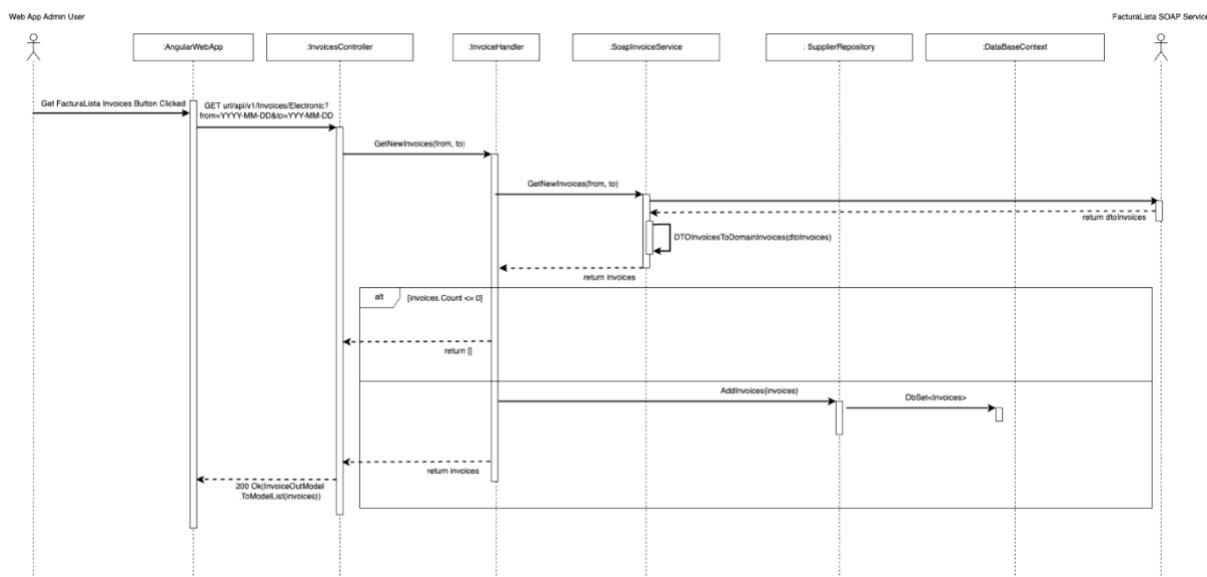


Ilustración 11 - Diagrama de secuencia de obtención de facturas

5.3. Plataformas tecnológicas y justificaciones de diseño

5.3.1. Elección de tecnologías y justificación

5.3.1.1. *Frontend web*

Angular y Flutter Web ofrecen enfoques distintos para el desarrollo de aplicaciones *web*, pero hay varias razones por las que el equipo podría elegir Angular sobre Flutter Web, especialmente en contextos empresariales.

Primeramente, Angular es un *framework* maduro basado en el lenguaje TypeScript, el cual ofrece un ecosistema robusto, incluyendo herramientas avanzadas de desarrollo y una muy amplia comunidad de soporte. Además, Angular está diseñado con un enfoque en el desarrollo de aplicaciones de página única (SPA), lo que lo hace ideal para proyectos que requieren una experiencia de usuario fluida y dinámica sin recargas de página.

Por otro lado, Flutter Web, siendo una extensión relativamente nueva de Flutter (originalmente enfocado para el desarrollo móvil), todavía está madurando en términos de rendimiento y compatibilidad con el navegador comparado con soluciones específicas de *web* como Angular.

Por estas razones, Angular puede ser la opción preferida para proyectos *web* complejos que buscan escalabilidad, mantenibilidad y una amplia gama de funcionalidades.

Por otro lado, el equipo cuenta con otro *framework* en la mira, el cual compite directamente con Angular, y se llama React. El uso de este *framework* es realmente masivo, y a la par de su mencionado competidor.

Tanto Angular como React, tienen un nivel similar de madurez en la industria, por lo cual, la decisión de cual elegir de los dos, no es fácil.

Una consideración muy importante, es que el equipo planea continuar con el desarrollo del sistema, y, por ende, al pensar a largo plazo, es preferible una tecnología con mayor consolidación en la industria, como lo son Angular y React.

Para lograr una mejor comparativa, y finalmente poder salir de dudas, el equipo realizó una tabla, donde se puntúan seis atributos, con un peso del uno al diez.

El resultado de la comparativa objetiva (Ilustración 12), dejó en evidencia que React parecería ser la opción más correcta si tenemos en cuenta los atributos evaluados.

Ponderación de atributos de tecnologías	Peso	Flutter Web	Angular	React
(1) - Experiencia del equipo	3	0	1	0
(2) - Popularidad y comunidad	2	0	1	1
(3) - Rendimiento	2	0	1	1
(4) - Facilidad de aprendizaje	2	1	0	1
(5) - Documentación y soporte	3	0	1	1
(6) - Ecosistema	2	1	0	1
(7) - Total promedio	-	4	10	11

Ilustración 12 - Comparativa de tecnologías *frontend*

Una vez aclarado el panorama, el equipo se decidió por Angular, como *framework* para el desarrollo de la aplicación *web*. Pese a obtener menos puntaje en la tabla de ponderación (Ilustración 12) de *frameworks*, lo que inclinó la balanza fue la experiencia previa de cada uno de los integrantes, por lo cual, no se requeriría que aprendan el *framework*. Además, el resultado de la tabla, tanto para React como para Angular, fueron similares.

El cliente consumirá los servicios del *backend* por medio de un navegador *web*, el cual viene instalado por defecto en cualquier celular o computadora.

La aplicación *web* se desarrolla en el editor de código Visual Studio Code, debido a la gran comunidad de este, y la facilidad para integrar *plugins* y funcionalidades de terceros.

Para simplificar el desarrollo, el equipo decidió utilizar un *template* para el *frontend* de la aplicación *web*. El mismo le provee al equipo una interfaz gráfica atractiva y moderna, con una disposición de menús y navegación que le facilita y agiliza el desarrollo al equipo, eliminando la necesidad de tener que codificar componentes ya existentes.

La alternativa a la utilización del *template*, era un desarrollo del producto desde cero. Debido a que los plazos del proyecto no son muy extensos, se decidió acortar camino por medio del uso de este.

El *template* utilizado, es gratuito y se puede encontrar en el siguiente [enlace](#).

5.3.1.2. *Frontend móvil*

Aquí el equipo discutió acerca de cuál era el mejor camino para tomar. Por un lado, contamos con Francisco Cabanillas, quien trabaja desarrollando *software* nativo de iOS con el lenguaje Swift, y

por otro lado contamos con Martín Sosa que trabaja desarrollando *software* móvil multiplataforma con la tecnología Flutter.

Andrés Quintero actualmente no está empleado dentro de la industria, pero recientemente trabajó junto a Martín, en el mismo puesto. Por ende, tenemos dos desarrolladores con experiencia en desarrollo móvil multiplataforma.

Al tener una ventaja de 2v1 respecto a la cantidad de desarrolladores por tecnología, es que se decidió por desarrollar la aplicación que utilizará la tablet del contenedor de alimentos en Flutter. De esta manera, ese desarrollo podría ser encarado en menos tiempo.

Otro motivo para reafirmar esta elección es que, si el día de mañana el equipo cuenta con otro cliente, se puede hacer una versión rápidamente para iOS, ya que Flutter permite compilar versiones para ambos sistemas operativos, con un solo código fuente.

También el factor económico inclinó la balanza hacia Flutter, debido a que la aplicación será utilizada en una tablet Android, las cuales son más baratas que un iPad.

La aplicación móvil se desarrolla en el entorno de desarrollo Android Studio, debido a la gran comunidad de este, la facilidad para integrar *plugins* y funcionalidades de terceros, y sobre todo porque es la recomendación oficial de Google (creador de Flutter).

5.3.1.3. API

.NET y Java Spring son dos *frameworks* muy populares para el desarrollo de aplicaciones *web*. La elección entre estos dos puede depender de varios factores, incluyendo la experiencia del equipo, el tipo del proyecto, y las preferencias de tecnología del equipo. Para un mayor contexto sobre lo que representa .NET y Java Spring, se debe revisar el siguiente anexo [.NET](#) y el [Java Spring](#).

Dado que todos los integrantes del equipo tienen experiencia en .NET debido al trabajo o la universidad, y sólo un miembro del equipo ha trabajado con Java Spring, la tabla de evaluación (Ilustración 13) puede reflejar estos puntos fuertes identificados en cada uno de los *frameworks* a analizar:

Ponderación de atributos de tecnologías	Peso	.NET	Java Spring
(1) - Experiencia del equipo	3	1	0
(2) - Popularidad y comunidad	2	1	1
(3) - Rendimiento	2	1	1
(4) - Facilidad de aprendizaje	2	1	1
(5) - Documentación y soporte	3	1	1

(6) - Ecosistema	2	1	1
(7) - Total promedio	-	15	12

Ilustración 13 - Comparativa tecnologías *backend*

Una vez culminada la etapa de ponderación de atributos de los *frameworks*, el equipo se decidió por .NET (en su versión 8.0) como *framework* para el desarrollo de la aplicación *web*. Aunque Java Spring es un competidor fuerte, la familiaridad extensa con .NET en el equipo y su integración nativa con otras tecnologías de Microsoft hacen que sea la opción más lógica y estratégica para el desarrollo del proyecto, garantizando una curva de aprendizaje casi inexistente para el equipo y un aprovechamiento máximo de las habilidades existentes.

La *API* se desarrolla en el entorno de desarrollo Rider. Como Rider es desarrollado por Jetbrains (al igual que Android Studio), el equipo consideró que era la opción ideal, ya que al contar con experiencia en el mercado profesional trabajando con Android Studio (por parte de dos integrantes), la confianza en dicha empresa es alta. Otra razón importante de la elección de Rider, es que como los tres integrantes del equipo desarrollan en Mac, Visual Studio dejaría de servir, ya que Microsoft le quitó el soporte el 31 de agosto de 2024. También es importante destacar, que el equipo cuenta con licencia estudiantil para acceder a dicho *software*, sin necesidad de abonar una suscripción.

5.3.1.4. *Machine learning*

Python y R son pilares en el análisis de datos y el desarrollo de modelos de *machine learning*. Sin embargo, existen diferencias significativas en cuanto a facilidad de uso, integración con otras herramientas y el soporte para operaciones a gran escala que pueden influir en la decisión de un equipo de desarrollo.

La experiencia del equipo en Python es considerable, dado que todos los integrantes han utilizado este lenguaje en cursos y proyectos anteriores de *machine learning*. Esto reduce significativamente la curva de aprendizaje y aumenta la productividad desde el inicio del proyecto. Además, Python cuenta con una gran popularidad y presencia dominante en diversas industrias, lo que se refleja en una mejor disponibilidad de recursos de aprendizaje y soporte técnico.

En términos de bibliotecas disponibles, Python tiene una ventaja con herramientas específicas como Scikit-learn, que son esenciales para el desarrollo de proyectos innovadores en inteligencia artificial. La facilidad de integración de Python con otras plataformas y tecnologías también es crucial para la implementación en producción y para operaciones a gran escala, lo que lo hace más adecuado para proyectos complejos y multidisciplinarios.

Tanto Python como R están bien documentados y cuentan con comunidades activas que contribuyen regularmente con nuevas soluciones y correcciones. Sin embargo, Python se actualiza más frecuentemente y su comunidad es más amplia. En cuanto a la flexibilidad en el manejo de datos, Python, con herramientas como Pandas y NumPy, ofrece capacidades superiores para el manejo y análisis de grandes volúmenes de datos, un aspecto crucial para el éxito en proyectos modernos de ciencia de datos.

Dadas estas consideraciones (Ilustración 14), Python es la elección preferida para el equipo, garantizando eficiencia y minimizando los riesgos en futuros proyectos de *machine learning*. Este enfoque asegura que el equipo pueda aprovechar al máximo sus habilidades previas y la vasta infraestructura de soporte que Python ofrece.

Ponderación de atributos de tecnologías	Peso	Python	R
(1) - Experiencia del equipo	3	1	1
(2) - Popularidad y comunidad	2	1	1
(3) - Rendimiento	2	1	1
(4) - Facilidad de aprendizaje	2	1	0
(5) - Documentación y soporte	3	1	1
(6) - Ecosistema	2	1	0
(7) - Total promedio	-	14	10

Ilustración 14- Comparativa tecnologías de *machine learning*

5.3.1.5. Base de datos

El equipo decidió que el motor de base de datos a utilizar sea PostgreSQL, por una sencilla razón. Uno de los integrantes del equipo, utiliza en su trabajo este motor, casi que en su día a día. Entonces, el mismo se ofreció a crear dicha base de datos, configurarla y enlazarla a la *API* de .NET por medio de los comandos "Add" (creación de las migraciones) y "Update" (actualización de la base de datos) del *ORM* (*Object Relational Mapping*) Entity Framework Core. En el siguiente anexo [ORM](#) hay mayor contexto sobre el *ORM*.

No se ponderaron atributos de comparación con otras tecnologías ya que el resto de los integrantes confió en quien propuso el uso de este motor de base de datos.

Otras dos importantes razones de peso son: 1) Entity Framework Core (el *ORM* utilizado por el equipo, tiene compatibilidad con PostgreSQL. 2) PostgreSQL se encuentra disponible para ser utilizado en la infraestructura de Azure.

Por otro lado, las entidades de negocio que se almacenan tienen amplias dependencias y vínculos entre sí (restricciones de integridad referencial). Eso implicó la necesidad del uso de un esquema relacional, en lugar de uno no relacional.

5.3.1.6. Infraestructura de TI

La experiencia de los integrantes del equipo en lo que corresponde a *DevOps*, siempre transcurrió por medio de Azure *DevOps*. Otra gran alternativa es AWS de la Empresa Amazon. Por la familiaridad del equipo con el ambiente de servicios en la nube de Microsoft, es que se decidió ir en esa dirección.

Por una razón económica, se le presentaron ambas propuestas al cliente del equipo. Sin embargo, los precios eran similares, entonces el cliente dio la posibilidad de que los integrantes del equipo decidan acerca de cuál tecnología utilizar.

5.4. Arquitectura interna de cada componente

5.4.1. Frontend

5.4.1.1. Cliente web Angular

5.4.1.1.1. Comunicación con el backend

La comunicación entre el cliente *web* Angular y el *backend* se maneja principalmente a través de servicios HTTP utilizando la biblioteca HttpClient de Angular. Esto permite enviar peticiones y recibir respuestas de manera asíncrona, asegurando que la interfaz de usuario no se bloquee mientras se espera la respuesta del servidor. Los interceptores se configuran para agregar automáticamente cabeceras comunes a cada solicitud, como los tokens de autenticación, lo que simplifica la gestión de peticiones seguras.

Además, se ha implementado la capacidad de definir tiempos de espera y cancelar solicitudes obsoletas, algo crucial para mejorar la eficiencia y la experiencia del usuario. Esto es especialmente útil cuando los usuarios cambian de vista o realizan múltiples acciones en un corto período, evitando que las solicitudes innecesarias continúen ejecutándose en segundo plano y utilizando recursos del sistema.

5.4.1.1.2. Patrones y uso de estándares

Reutilización de componentes

En el cliente *web* Angular, se siguen varios patrones y principios de diseño con el objetivo de mantener el código modular y fácilmente mantenible. Una práctica clave ha sido la reutilización de componentes en toda la aplicación. En lugar de duplicar código y lógica en diferentes vistas, se

han creado componentes genéricos que pueden adaptarse y utilizarse en múltiples lugares, lo que reduce la redundancia de código y facilita la realización de cambios en un solo punto.

Componentes y servicios

La arquitectura se basa en el patrón de Componentes y Servicios, donde los componentes manejan únicamente la presentación y la interacción del usuario, y los servicios se encargan de la lógica de negocio y las llamadas al *backend*. Esta separación mejora la claridad y organización del código, permitiendo que los desarrolladores trabajen en áreas específicas de la aplicación sin interferir con otras partes.

Guards para protección de rutas

Otra característica importante es el uso de guards para proteger las rutas de la aplicación. Los guards controlan el acceso a diferentes secciones del sistema, permitiendo que solo usuarios con ciertos permisos o condiciones específicas puedan navegar a determinadas áreas. Esto es fundamental para la seguridad, ya que garantiza que los usuarios no accedan a secciones que no deberían ver, además de mejorar la experiencia de usuario al redirigirlos adecuadamente en caso de restricciones.

Uso del lenguaje de programación TypeScript y modularidad

El uso de TypeScript con tipado estricto asegura la consistencia y previene errores comunes durante el desarrollo, al mismo tiempo que facilita la colaboración entre equipos. La organización modular del código, junto con el uso de patrones y estándares recomendados por Angular, asegura que la aplicación pueda ser escalable, fácil de mantener y preparada para el futuro.

Directorios principales

La estructuración de los directorios (Ilustración 15) tomada por el equipo, se compone de la siguiente manera:

La carpeta *layouts/admin-layout* contiene el código relacionado con la estructura visual y de presentación del diseño administrativo de la aplicación. Aquí se incluyen archivos que manejan la configuración de la interfaz de usuario, como el HTML, los estilos en SCSS, el componente en TypeScript que define la lógica, y los módulos y rutas que agrupan y configuran cómo este layout es utilizado dentro de la aplicación.

En *models/in* se agrupan los modelos de datos que representan entidades específicas dentro de la aplicación. Estos modelos incluyen clases que definen cómo se estructuran los datos que la aplicación recibe o envía, como "*product-type*", "*supplier*", "*user*", y "*credentials*", etc. Estos modelos son esenciales para la comunicación con el *backend* y ayudan a estructurar los datos de forma clara y consistente.

La carpeta *pages* contiene las diferentes vistas o páginas que componen la interfaz de usuario de la aplicación. Aquí están organizadas por funcionalidad, como el tablero, *login*, notificaciones, tipos de productos, proveedores, y más. Cada subcarpeta contiene los archivos específicos para cada vista, que definen la lógica y el diseño de esa página en particular. Estas páginas interactúan con los servicios y modelos para mostrar la información relevante al usuario.

En *services* se ubican los archivos que manejan la lógica de negocio relacionada con la comunicación entre la aplicación y las APIs externas. Los servicios incluyen funcionalidades como la gestión de productos, proveedores, usuarios, etc, además de la implementación de guards de sesión que controlan el acceso a rutas específicas. Cada servicio encapsula las operaciones que interactúan con el *backend*, proporcionando una abstracción para simplificar el manejo de datos y asegurar consistencia en todas las partes de la aplicación.

La carpeta *shared* contiene componentes que json reutilizables en diferentes partes de la aplicación, como el navbar, el footer, y componentes genéricos como formularios y vistas. Estos componentes permiten estandarizar elementos comunes en toda la aplicación, facilitando la reutilización del código y manteniendo una consistencia visual y funcional en las diferentes páginas.

Finalmente, la carpeta *sidebar* contiene los componentes relacionados con la barra lateral de navegación de la aplicación. Esta carpeta incluye el HTML y la lógica en TypeScript para el componente de la barra lateral, que gestiona las opciones de navegación y enlaces a diferentes secciones de la aplicación.

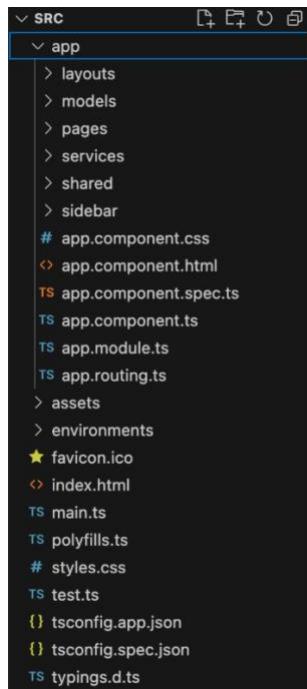


Ilustración 15 - Ejemplo de estructura de carpetas del proyecto Angular del obligatorio (en construcción)

5.4.1.2. Cliente móvil Flutter

5.4.1.2.1. Patrones y uso de estándares

Manejo de estados

En una aplicación Flutter, el manejo de estados es crucial para mantener la UI sincronizada con los datos correspondientes a mostrar. Se utilizan diversas arquitecturas y bibliotecas para gestionar el estado de manera eficiente. Una de las más populares es BLoC (Business Logic Component). Este patrón ayuda a separar la lógica de negocio de la interfaz de usuario, gestionando el flujo de datos a través de "events" y "states". Esto no solo facilita la prueba y el mantenimiento del código, sino que también promueve la reutilización de la lógica de negocio.

Clean Architecture

Es un enfoque de diseño que promueve la separación de preocupaciones y facilita el mantenimiento y la escalabilidad del código. Esta arquitectura (Ilustración 16) divide la aplicación en varias capas bien definidas, asegurando que la lógica de negocio esté aislada de los detalles de implementación como *frameworks*, bibliotecas de UI y detalles de infraestructura. Esto se logra mediante la organización del código en diferentes capas, cada una con responsabilidades específicas, lo que permite que las modificaciones en una capa no afecten innecesariamente a las demás.

En la capa de datos (Data Layer), se maneja la obtención y almacenamiento de datos, ya sea desde *APIs* externas, bases de datos locales o cualquier otra fuente de datos. Esta capa incluye componentes como DTOs (Data Transfer Objects), que definen cómo se estructuran los datos transferidos entre la fuente de datos y la aplicación. También incluye repositorios que actúan como intermediarios entre las fuentes de datos y las otras capas de la aplicación, proporcionando una abstracción de la fuente de datos y simplificando la interacción con la capa de dominio.

La capa de dominio (Domain Layer) contiene la lógica de negocio central de la aplicación. Aquí se definen las entidades de dominio, que representan los conceptos clave del negocio, y los casos de uso, que contienen las reglas de negocio y las operaciones que pueden realizarse. Esta capa es completamente independiente de cualquier *framework* o tecnología de *frontend* o *backend*, lo que la hace altamente reutilizable y testeable. Al estar centrada únicamente en la lógica de negocio, permite que cualquier cambio en los requerimientos del negocio se maneje de manera aislada, sin impacto en las capas de presentación o datos.

Por último, la capa de presentación (Presentation Layer) se encarga de la interfaz de usuario y la interacción con el usuario. Esta capa incluye patrones como BLoC (Business Logic Component), que manejan la lógica de presentación de manera reactiva, transformando eventos del usuario en estados que la UI puede consumir. Además, contiene las vistas o páginas de la aplicación, que

observan los estados emitidos por los BLoCs o cualquier otra solución de manejo de estado, para reflejar los cambios en la interfaz de usuario. Esta estructura asegura una experiencia de usuario consistente y facilita la prueba unitaria y de integración, mejorando así la calidad del *software*.

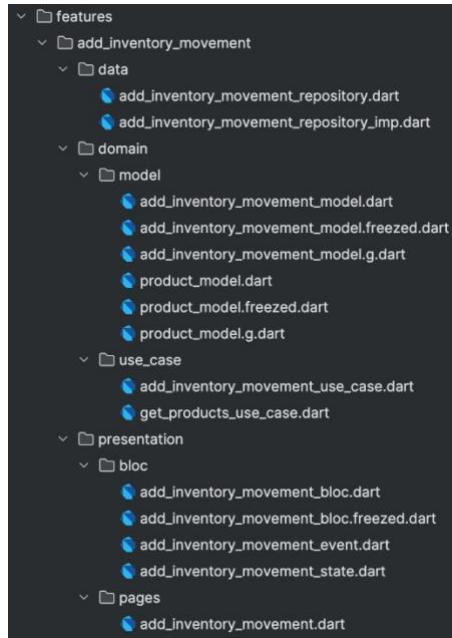


Ilustración 16 - Estructura de una *feature* Clean Architecture en Flutter

Rutas y navegación

La gestión de rutas y navegación en Flutter es fundamental para la experiencia de usuario, permitiendo una navegación fluida y consistente a través de diferentes pantallas. El equipo utilizó GoRouter, la cual es una biblioteca que simplifica este proceso mediante una configuración declarativa de rutas. Soporta la navegación basada en URLs, incluyendo deep linking, lo cual es esencial para integraciones con navegadores y para proporcionar una experiencia coherente en diferentes plataformas. GoRouter también facilita la protección de rutas y la gestión de parámetros de manera estructurada.

Generación de código

La generación de código en Flutter ayuda a automatizar tareas repetitivas y a garantizar consistencia y precisión en el código. Freezed es una herramienta popular para este propósito, especialmente en la creación de clases inmutables y modelos de datos. Freezed es una librería que genera automáticamente métodos en clases (como por ejemplo métodos de conversión a JSON o desde JSON), lo que reduce la carga de escribir y mantener estas implementaciones manualmente. Esto no sólo acelera el desarrollo, sino que también minimiza errores, asegurando que los datos sean manejados de manera segura e inmutable.

El objetivo de la arquitectura seleccionada para la aplicación Flutter, es que sea un producto profesional, mantenable, y actualizado con los más modernos estándares de la industria.

5.4.1.2.2. Comunicación con el *backend*

La comunicación con el *backend* se realiza por medio de una librería llamada Dio. Dio es una librería flexible para realizar peticiones HTTP en aplicaciones Flutter. Proporciona características avanzadas como interceptores, cancelación de peticiones, solicitudes con tiempo límite, manejo automático de errores y más. Estas características hacen que la gestión de peticiones HTTP y la manipulación de respuestas sean mucho más eficientes y manejables.

Dio permite configurar interceptores que pueden modificar las solicitudes y respuestas de manera centralizada. Por ejemplo, se pueden agregar encabezados comunes como tokens de autenticación a todas las peticiones sin necesidad de hacerlo manualmente en cada solicitud. También es posible manejar respuestas globales, como la renovación automática de tokens expirados, lo que simplifica la gestión de autenticación.

Además, Dio facilita la configuración de tiempo límite para solicitudes, lo cual es útil para manejar situaciones en las que el servidor no responde o la conexión es lenta, evitando así que la aplicación se quede esperando indefinidamente. La capacidad de cancelar solicitudes también es importante, especialmente en escenarios donde una solicitud puede quedar obsoleta, como cuando un usuario navega a otra pantalla mientras una petición aún está en curso.

El equipo decidió utilizar Dio como una herramienta que simplifica y mejora la interacción con servicios *backend* en la aplicación móvil Flutter, proporcionando una forma limpia y eficaz de manejar la comunicación de red, lo que contribuye a una mejor experiencia de usuario y una aplicación más robusta. De esta forma, el equipo no tiene que realizar peticiones HTTP crudas, sino que aprovecha los estándares definidos por esta librería (ahormando tiempo).

5.4.2. Backend

5.4.2.1. API

5.4.2.1.1. Modelo de tres Capas

El modelo de arquitectura que organiza una aplicación en tres capas distintas, conocidas como la capa de presentación, la capa de negocio, y la capa de datos, es conocido como "arquitectura de tres capas". Este enfoque arquitectónico es popular en el desarrollo de *software* debido a su capacidad para separar las responsabilidades, lo que promueve la modularidad y la reutilización del código. En el proyecto, se aplica este modelo de tres capas dentro de cada submódulo del *backend*.

La capa de presentación, también conocida como los controladores de la *API*, es responsable de la interacción con el usuario final. Esta capa maneja las solicitudes provenientes del *frontend*, facilitando la comunicación con el usuario y gestionando la navegación y presentación de la información. Su función es actuar como un intermediario entre el usuario y la lógica de negocio, asegurando que la interfaz sea amigable y eficiente.

La capa de negocio contiene la lógica central del sistema. En esta capa se realizan todos los procesos de negocios, se aplican las reglas comerciales y se gestionan las operaciones específicas del sistema. La independencia de esta capa respecto a la presentación y los datos permite una mayor facilidad para realizar pruebas unitarias, además de facilitar la reutilización del código y la implementación de cambios en la lógica de negocio sin afectar otras partes del sistema.

Por último, la capa de datos es la encargada de la gestión y el acceso a los datos. Incluye la interacción con las bases de datos, el almacenamiento de datos y la conexión con servicios externos para la persistencia de la información. Esta separación permite que los cambios en la tecnología de almacenamiento de datos no afecten la lógica de negocio ni la presentación, proporcionando flexibilidad y mantenimiento simplificado.

5.4.2.1.2. Módulos

BusinessLogic

El módulo BusinessLogic implementa y coordina la lógica de negocio del sistema, abarcando procesos fundamentales como la gestión de usuarios, productos, tipos de productos, proveedores, movimientos de inventario, eventos de restaurante, facturas, habitaciones, entradas de huéspedes, reportes, etc.

Este módulo garantiza que todas las operaciones se ejecuten de acuerdo con las reglas de negocio definidas por el cliente, manteniendo la integridad y consistencia de los datos.

En la gestión de usuarios, el módulo cubre desde la creación y administración de usuarios hasta su autenticación, validando roles y permisos para asegurar el cumplimiento de las políticas de seguridad del sistema.

La gestión de productos incluye la creación, actualización y eliminación de productos, aplicando reglas como la validación de unicidad de nombres y la correcta asignación a categorías y proveedores. También maneja alertas de stock bajo, notificando a los usuarios responsables mediante correo electrónico cuando un producto alcanza su umbral mínimo.

Para los proveedores, se valida que los datos ingresados, como el nombre y el RUT, sean correctos y consistentes, asegurando que no existan duplicados en el sistema antes de realizar operaciones de registro o actualización.

En cuanto a los movimientos de inventario, el módulo se encarga de registrar entradas y salidas de productos, así como de gestionar el control de stock, asegurando que los niveles se mantengan actualizados. También aplica reglas para movimientos específicos como compras, consumos, vencimientos y roturas.

La gestión de eventos de restaurante garantiza que los eventos se ajusten a las entradas de huéspedes, validando fechas, tipos de eventos y la disponibilidad de habitaciones habilitadas. El sistema previene duplicados en los registros de eventos.

Para la gestión de facturas, se validan los datos antes de su registro (ay sea manual o automático con la integración con FacturaLista), asegurando que las fechas y los proveedores sean correctos, y garantizando que no existan inconsistencias entre las fechas de emisión y vencimiento. Se previenen duplicados mediante la utilización de la serie y número de factura como identificadores.

También el módulo gestiona lo que corresponde a la generación de los reportes/dashboards que serán utilizados desde el *frontend web*.

Finalmente, el módulo permite la gestión de habitaciones y entradas de huéspedes, validando que las habitaciones asignadas estén habilitadas y que las entradas respeten las fechas de alojamiento, asegurando que todos los registros sean consistentes y completos.

Este módulo está diseñado para ser extensible, permitiendo la integración de nuevas funcionalidades en el futuro según las necesidades del cliente, garantizando una base sólida para el crecimiento del sistema.

ExternalServices

Este módulo agrupa las clases e interfaces necesarias para la comunicación con los servidores de la empresa de facturación electrónica Facturalista.

Para gestionar la comunicación entre la *API* y FacturaLista, se aplicó el patrón *adapter*.

¿Por qué? No solo que el desacoplamiento se ve favorecido debido al uso de la interfaz del adaptador, la cual hace que quien la consuma no conozca la implementación, sino que permite que el equipo pueda tener más de una implementación del método que obtiene las facturas.

FacturaLista tiene un servicio *REST* inestable hoy en día, pero que el equipo sabe que será estable a mediano plazo. Por ende, y para disminuir el impacto de cambio en el sistema, el equipo decidió por implementar el patrón *adapter*. De esta manera, la clase que consume la interfaz, jamás se entera con qué implementación está trabajando (gracias a la inyección de dependencias). Inicialmente se cuenta con la implementación de obtención de facturas, por medio del servicio *SOAP*.

En un futuro (año 2025), el equipo hará la transición, creando una implementación basada en el servicio *REST*. El único cambio que deberá hacer el equipo en la *API* es en la sección de inyección de dependencias. Se debe ingresar el nombre de la clase concreta de la nueva implementación, y el problema queda solucionado.

Como los integrantes del equipo plantearon la lógica de esta manera, el código que consume la interfaz, jamás deberá de ser modificado al cambiar de implementación (a no ser que se modifique alguna regla de negocio).

CC_Manager_services.Controllers

El módulo `CC_Manager_services.Controllers` es responsable de gestionar las interacciones entre el *frontend* y la lógica de negocio, facilitando la comunicación entre los usuarios y las funcionalidades del sistema.

Cada controlador en este módulo está diseñado para manejar solicitudes HTTP relacionadas con la administración de monedas, movimientos de inventario, facturas, productos, tipos de productos, proveedores, usuarios, eventos de restaurante, habitaciones, entradas de huéspedes, facturas, reportes, etc. Actúan como el punto de entrada de las solicitudes de los usuarios, asegurando que sean procesadas correctamente y que se comuniquen con la capa de negocio de manera eficiente.

En la gestión de usuarios, el controlador correspondiente permite agregar, actualizar, eliminar y obtener detalles de usuarios, además de gestionar permisos específicos, garantizando que el sistema maneje adecuadamente roles y accesos. Para los proveedores, el controlador facilita la adición, actualización y obtención de información de proveedores, validando que no existan duplicados y que los datos sean correctos antes de realizar operaciones.

La gestión de productos y tipos de productos se centraliza en controladores que permiten agregar, actualizar y consultar productos, asegurando que las operaciones respeten las reglas de negocio, como la unicidad de nombres y la correcta asociación con proveedores y tipos de productos. Los controladores también manejan movimientos de inventario, registrando entradas y salidas de productos, y asegurando que los niveles de stock se mantengan actualizados.

Para la gestión de eventos de restaurante, el controlador garantiza que se respeten las reglas de negocio al registrar eventos relacionados con las entradas de huéspedes, validando fechas, tipos de eventos y disponibilidad de habitaciones. Asimismo, en la administración de facturas, los controladores permiten a los usuarios consultar, agregar y actualizar facturas, ofreciendo la posibilidad de ordenarlas por diferentes criterios y acceder a detalles específicos como montos pendientes.

También el módulo cuenta con el correspondiente controlador capaz de gestionar lo que corresponde a la generación de los reportes/*dashboards*.

Finalmente, el módulo asegura una comunicación fluida entre el *frontend* y la lógica de negocio, garantizando que todas las solicitudes de los usuarios sean procesadas con eficiencia y seguridad, y facilitando la integración de nuevas funcionalidades a medida que el sistema evoluciona.

CC_Manager_services.Filters

El paquete CC_Manager_services.Filters es responsable de la gestión de excepciones y la implementación de políticas de seguridad dentro del sistema. Su función principal es interceptar, manejar errores que ocurren durante las solicitudes al servidor, proporcionar respuestas adecuadas al cliente y registrar los detalles de las excepciones para análisis y resolución.

Este módulo captura una amplia variedad de excepciones, como errores de credenciales, problemas con la base de datos, excepciones de validación y errores en la comunicación con la API, asegurando que cada excepción tenga un manejo adecuado con códigos de estado HTTP apropiados y mensajes claros.

Además de gestionar excepciones, este paquete también se encarga de registrar los detalles de las excepciones en la base de datos, incluyendo el tipo de excepción, el mensaje de error, la traza de pila, los parámetros de la solicitud y los datos del usuario que generó el error, en caso de estar autenticado. Esto facilita el análisis de fallos, mejora el proceso de resolución de problemas y garantiza la estabilidad del sistema.

En términos de seguridad, el paquete también implementa el manejo de autenticación y autorización, verificando los tokens JWT para validar que solo los usuarios con los permisos adecuados puedan acceder a los recursos protegidos. A través de políticas de autorización personalizadas, el sistema asegura que los usuarios solo interactúen con las funcionalidades a las que tienen acceso, protegiendo la confidencialidad e integridad de los datos.

Domain

El módulo Domain define las entidades clave y sus relaciones dentro del sistema, actuando como la capa de modelo de datos que respalda la lógica de negocio. Este módulo contiene las estructuras de datos que representan los elementos esenciales del negocio, como usuarios, productos, proveedores, facturas, movimientos de inventario, huéspedes, habitaciones, reportes, etc. Cada entidad incluye propiedades que describen sus características y relaciones con otras entidades, estableciendo una base sólida para la manipulación de datos y la implementación de reglas de negocio.

El módulo también contiene clases para la validación de excepciones y manejo de errores, como ExceptionLog, que registra detalles críticos cuando ocurre un error en el sistema, proporcionando información valiosa para el análisis posterior.

El diseño de este módulo Domain, permite una integración fluida con otros componentes del sistema, asegurando que las operaciones de negocio respeten las reglas definidas y que las relaciones entre las entidades estén correctamente estructuradas para la manipulación eficiente de los datos.

Domain.Enums

Este módulo agrupa las enumeraciones que definen categorías clave para la organización de los datos y procesos dentro del sistema. Estas enumeraciones permiten clasificar diversos aspectos del negocio, aportando coherencia y estructura a las operaciones.

MovementTypeEnum es esencial para gestionar los movimientos de inventario, categorizando las entradas y salidas de productos. Las categorías incluyen Income para registrar las entradas de productos, Exit para las salidas, Expiration para movimientos relacionados con productos vencidos y Uselessness para productos que han perdido su utilidad.

PaymentTypeEnum organiza las formas de pago contenidas en las facturas del hotel, distinguiendo entre pagos en efectivo (Cash), crédito (Credit), y (NotSpecified) una tercera opción para aquellas facturas donde no se especifica claramente.

Finalmente, RestaurantEventTypeEnum clasifica los turnos gastronómicos del comedor del hotel, dentro del sistema, como: Breakfast, Lunch, Snack y Dinner, asegurando una adecuada gestión de los servicios ofrecidos a los huéspedes en distintos momentos del día.

Domain.Exceptions

El namespace Domain.Exceptions en *ChefCheck-Manager* define una serie de excepciones personalizadas que representan errores específicos dentro de la aplicación, proporcionando una estructura clara y coherente para gestionar situaciones excepcionales. Este módulo está diseñado para mejorar la robustez del sistema y optimizar la experiencia del usuario al manejar errores de manera precisa y contextual.

La clase NotFoundException extiende de NullReferenceException y se utiliza cuando un recurso o entidad solicitada no puede ser encontrada en el sistema. Esta excepción permite generar un mensaje claro para el usuario, indicando específicamente qué tipo de objeto no fue localizado, lo que facilita la identificación del error y mejora la comunicación entre el sistema y el cliente.

La excepción FromToException extiende de ArgumentException y se utiliza para situaciones en las que hay una incoherencia en los parámetros de fecha, como cuando la fecha "desde" es mayor que la fecha "hasta". Esta clase permite manejar de manera eficiente los errores relacionados con rangos de tiempo, asegurando que las operaciones basadas en fechas se ejecuten correctamente.

Por otro lado, `EmptyIndexPageSizeException`, también derivada de `ArgumentException`, se emplea para validar los parámetros de paginación en las solicitudes. Esta excepción se lanza cuando el índice o el tamaño de página son valores no válidos, evitando así que el sistema procese solicitudes que podrían afectar la funcionalidad o generar errores inesperados.

En conjunto, este grupo de excepciones personalizadas garantiza que los errores comunes dentro del sistema sean manejados de manera clara y específica, mejorando tanto la estabilidad del sistema como la claridad en la comunicación de errores hacia los desarrolladores y usuarios finales.

Domain.Reports

Este paquete contiene las clases necesarias para dar formato a los reportes que verán los administradores en la aplicación *web*.

Interfaces.ExternalServices

Este módulo agrupa la interfaz que debe ser utilizada por cualquier implementación que tenga como objetivo, gestionar la comunicación con los servicios externos de terceros. Actualmente, y como ha sido mencionado anteriormente, solo contiene la definición para poder obtener las facturas mediante el servicio expuesto por `FacturaLista`.

Interfaces.Handlers

El módulo `Interfaces.Handlers` define un conjunto de interfaces que sirven como contratos para las operaciones de la capa de negocio. Estas interfaces especifican los métodos esenciales que deben ser implementados para gestionar las entidades clave del dominio del sistema. Al proporcionar una estructura clara para las operaciones de negocio, este módulo permite una separación efectiva de responsabilidades, facilitando la extensibilidad y el mantenimiento del sistema.

Entre las principales funcionalidades de este módulo se encuentran la gestión de proveedores (junto a la gestión de facturas), habitaciones, huéspedes y productos, lo que permite agregar, actualizar y consultar estas entidades siguiendo las reglas de negocio establecidas.

Las interfaces también permiten manejar facturas y movimientos de inventario, asegurando que las transacciones financieras y el control de stock se realicen de manera eficiente. Adicionalmente, se cubren aspectos como la reportería/dashboards y la gestión de eventos de restaurante, que permite organizar los servicios gastronómicos vinculados a las entradas de huéspedes, y la administración de entradas de huéspedes, esencial para gestionar la ocupación en el hotel.

El módulo también aborda la gestión de monedas, facilitando la administración de divisas en el sistema, y la implementación de servicios de correo electrónico para notificar sobre situaciones críticas, como alertas de stock bajo. Estas interfaces aseguran que la lógica de negocio se

implemente de manera coherente, siguiendo un patrón estructurado que favorece la escalabilidad y robustez del sistema.

Interfaces.Repositories

El módulo Interfaces.Repositories establece los contratos para el acceso y manipulación de los datos persistentes del sistema. Estas interfaces definen los métodos esenciales que deben implementarse para interactuar con las entidades del dominio y proporcionar una abstracción clara entre la lógica de negocio y la capa de persistencia.

Cada interfaz se centra en una entidad específica del sistema, permitiendo la adición, actualización, eliminación y consulta de datos, garantizando que las operaciones de acceso a la base de datos sean consistentes y eficaces.

Este conjunto de interfaces proporciona una estructura clara y mantenible para el acceso a los datos, asegurando que la capa de persistencia cumpla con los requisitos del negocio, y permite que las entidades del sistema se gestionen de manera eficiente y extensible.

Models.Input

Models.Input (también conocido como data transfer object de entrada) se encarga de definir modelos que representan los datos de entrada necesarios para realizar operaciones en el sistema. Estos modelos se utilizan para recibir y procesar la información proporcionada por los usuarios a través de la *API*.

Cada modelo de entrada incluye propiedades que corresponden a los datos que el sistema necesita para realizar una acción específica como: agregar, actualizar o eliminar información en la base de datos. Además, estos modelos suelen incluir métodos para convertir los datos recibidos en objetos que pueden ser utilizados por la lógica de negocio, facilitando la integración y manipulación de datos de manera consistente y segura.

Models.Output

Models.Output (también conocido como data transfer object de salida) se encarga de definir los modelos de salida que representan la información procesada que se envía al usuario o cliente del sistema. Estos modelos toman los datos internos del sistema y los transforman en un formato adecuado para su visualización o para ser consumidos por otras aplicaciones.

Los modelos de salida incluyen propiedades que reflejan los datos necesarios para mostrar información relevante y útil al usuario, como detalles de productos, movimientos de inventario, facturas, entre otros. Además, estos modelos incluyen métodos para convertir listas de objetos internos en listas de modelos de salida, facilitando la presentación de datos de manera consistente

y estructurada. Esto asegura que la información se entregue de manera clara y accesible, mejorando la experiencia del usuario y la integridad de los datos presentados.

Repository

En el paquete Repository, el desacoplamiento entre el repositorio y la base de datos se logra a través del uso de Entity Framework Core (*EF Core*). *EF Core* actúa como un Object-Relational Mapper (*ORM*), permitiendo a los desarrolladores trabajar con bases de datos usando objetos .NET. Esto ofrece una abstracción de los detalles específicos de la base de datos, facilitando el trabajo con diferentes sistemas de gestión de bases de datos sin cambiar la lógica de negocio.

Al usar Entity Framework Core, el módulo Repository define las operaciones CRUD (crear, leer, actualizar, eliminar) a través de métodos genéricos y específicos de las entidades. Estas operaciones se implementan utilizando los contextos de datos de *EF Core*, que manejan la conexión a la base de datos y la ejecución de consultas SQL de forma transparente para los desarrolladores.

Los repositorios no necesitan conocer los detalles específicos de la base de datos, como la estructura de las tablas o el dialecto SQL. En su lugar, trabajan con modelos de dominio que representan las entidades de datos. *EF Core* se encarga de traducir estas operaciones de alto nivel en comandos SQL optimizados para la base de datos subyacente.

Este desacoplamiento permite cambiar el proveedor de base de datos (por ejemplo, de SQL Server a PostgreSQL) con nulas modificaciones en el código del repositorio, siempre y cuando se utilicen las características soportadas por *EF Core* en ambos sistemas de bases de datos. Además, facilita el mantenimiento y la prueba de la aplicación, ya que los desarrolladores pueden simular el comportamiento de la base de datos usando bases de datos en memoria o servicios de prueba durante el desarrollo y las pruebas.

Testing API

Testing API se encarga de realizar pruebas unitarias para verificar el correcto funcionamiento de los controladores de la *API*. Estas pruebas aseguran que las funcionalidades de la *API* respondan correctamente a diferentes escenarios y entradas.

Utilizando el marco de trabajo de pruebas unitarias, este módulo emplea objetos simulados (*mocks*) para reemplazar dependencias reales. Estos mocks son configurados para imitar el comportamiento de componentes externos, permitiendo probar el código en aislamiento. Por ejemplo, en lugar de interactuar con la base de datos real, los mocks permiten simular la interacción, verificando que los métodos sean llamados con los parámetros esperados y que las respuestas sean las adecuadas.

El módulo realiza pruebas en varias áreas clave de la *API*, como la adición, actualización y eliminación de elementos, y la recuperación de datos. Las pruebas verifican que los métodos de los controladores de la *API* devuelvan los resultados esperados, como códigos de estado HTTP y datos correctos. También aseguran que los errores y excepciones sean manejados adecuadamente, devolviendo respuestas coherentes con el manejo de errores definido en el sistema.

Ninguno de los paquetes de testing (cuatro en total), fueron diagramados (tanto dentro del diagrama de paquetes, como en los diagramas de clases), debido a que el equipo no lo consideró necesario, por la razón de que dichos paquetes no forman parte de la solución del negocio. El objetivo del diagrama de paquetes es visualizar cómo se relacionan unos con los otros en el contexto específico de este problema de negocio en particular.

El equipo sabe que en cualquier proyecto donde existan paquetes de testing, habrá dependencias hacia el dominio, lógica de negocio, repositorio, etc., manteniendo una estructura similar. Por eso el equipo no lo consideró como algo imprescindible.

Testing.BusinessLogic

Testing.BusinessLogic se enfoca en realizar pruebas unitarias para validar la lógica de negocios de la *API*. Su principal objetivo es asegurar que los componentes lógicos, funcionen correctamente bajo diversas condiciones.

Este módulo también utiliza mocks para reemplazar las implementaciones reales de repositorios y otros servicios externos. Esto permite que las pruebas se realicen en un entorno controlado, garantizando que se pruebe únicamente la lógica interna del sistema sin depender de componentes externos como bases de datos o servicios de autenticación.

Testing.DataAccess

Testing.DataAccess está diseñado para realizar pruebas unitarias de los componentes relacionados con el acceso a datos. Este módulo tiene el propósito de validar la funcionalidad de todos los distintos repositorios que interactúan con la base de datos, asegurando que las operaciones CRUD (crear, leer, actualizar y eliminar) se realicen correctamente y que se manejen adecuadamente las excepciones.

Testing.Filters

Testing.Filters se dedica a probar el filtro de manejo de excepciones dentro del sistema *ChefCheckManager*. Este filtro es responsable de interceptar y manejar las excepciones que ocurren durante la ejecución de la *API*, proporcionando respuestas adecuadas y detalladas a los usuarios finales (que ayuden a determinar la causa del error).

Las pruebas incluyen la verificación de respuestas ante diferentes tipos de excepciones, como errores de solicitudes incorrecta, recursos no encontrados, excepciones de referencia nula, argumentos no válidos, errores de conversión, archivos no encontrados, errores de JSON, fallos en la conexión con la base de datos, errores de concurrencia (también en la base de datos), etc.

En el siguiente diagrama (Ilustración 17), se puede observar una representación de la interacción de todos los módulos anteriormente descritos.

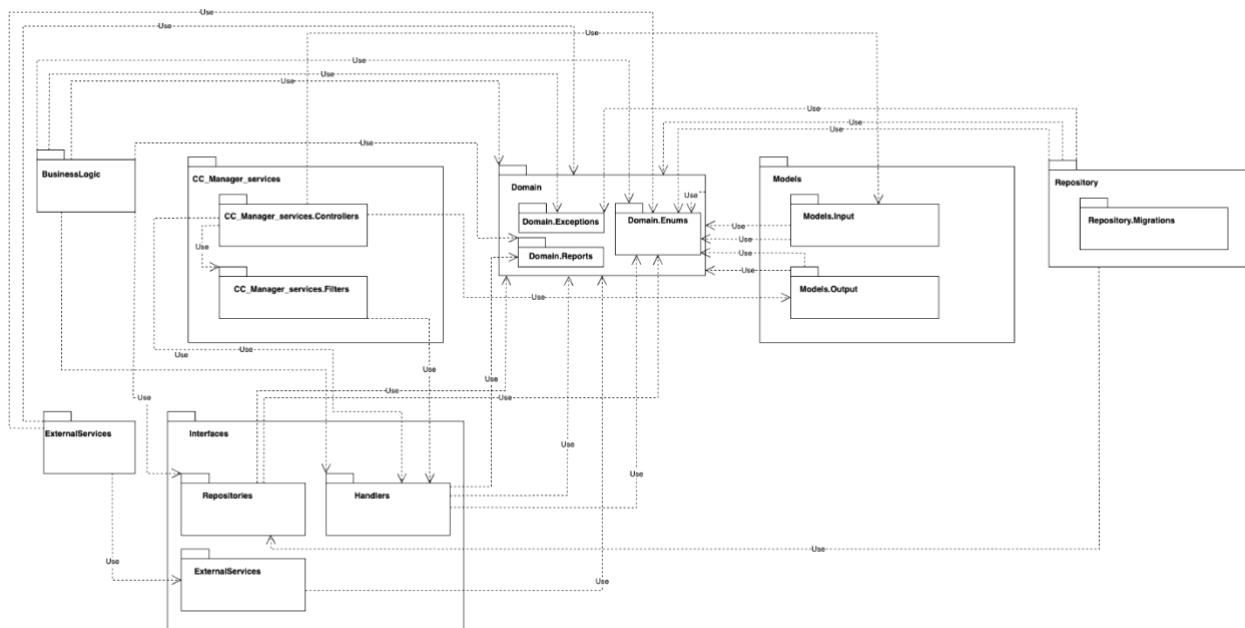


Ilustración 17 - Diagrama de paquetes

Los diagramas de clase (por paquete), se pueden visualizar en el anexo [Diagramas de clase \(por paquete\)](#).

5.4.2.1.3. Inyección de dependencias e inversión de dependencias

En el proyecto, la inyección de dependencias (DI) y el principio de inversión de dependencias (DIP) son esenciales para crear una arquitectura de *software* modular y fácilmente mantenible. Estos principios están bien implementados en el sistema (Ilustración 18) especialmente en la configuración del contenedor de servicios en el archivo "Program.cs".

La DI se utiliza para registrar servicios y repositorios necesarios para la aplicación. Por ejemplo, los servicios como "IProductHandler", "IIInvoiceHandler", "IAuthManagementHandler" e "IInventoryMovementHandler" se registran junto con sus implementaciones concretas. Estos servicios encapsulan la lógica de negocio y se inyectan en los controladores y otras partes del sistema donde se requieren. Los repositorios como "IProductRepository", "IIInvoiceRepository", "IAuthManagementRepository" e "IInventoryMovementRepository" también se registran para

manejar la interacción con la base de datos. Esto permite que los servicios no dependan directamente de una implementación específica de acceso a datos, facilitando el cambio de la tecnología de persistencia sin afectar otras partes del sistema.

El principio de inversión de dependencias se respeta al definir interfaces para cada servicio y repositorio, y luego inyectar estas interfaces en los consumidores de las dependencias, en lugar de depender directamente de implementaciones concretas. Esto no solo facilita el mantenimiento y la escalabilidad, sino que también mejora la capacidad de testear del sistema, permitiendo la inyección de implementaciones simuladas (gracias a la participación de mocks) durante las pruebas unitarias.

```
// Register services with the DI container
builder.Services.AddScoped<IProductHandler, ProductHandler>();
builder.Services.AddScoped<IInvoiceHandler, InvoiceHandler>();
```

Ilustración 18 - Ejemplo de inyección de dependencias

5.4.2.1.4. Estándares del *framework*

En el desarrollo de la *API* del proyecto, el equipo ha aprovechado diversas funcionalidades avanzadas del *framework* .NET, lo que ha permitido un desarrollo eficiente y profesional. Uno de los aspectos fundamentales ha sido el uso de ASP.NET *Core* para manejar la arquitectura de la aplicación, incluyendo la definición de controladores y acciones que permiten la gestión de rutas y métodos HTTP. Esto facilita la interacción con los recursos de la aplicación de manera estructurada y coherente.

El *framework* proporciona serialización y deserialización automática de JSON, lo que simplifica la comunicación entre el cliente y el servidor, asegurando que los datos se transfieran en un formato común y fácilmente interpretable. Este manejo automático de JSON es crucial para la eficiencia y la simplicidad en el intercambio de datos.

Para asegurar la calidad y fiabilidad del código, se integró un *framework* de testing unitario. Esto ha permitido al equipo realizar pruebas automáticas y rigurosas de las distintas funcionalidades de la aplicación, asegurando que cada componente funcione correctamente de manera aislada. Las pruebas unitarias son esenciales para detectar errores de manera temprana y garantizar que los cambios en el código no introduzcan nuevas fallas.

Además, se ha utilizado Swagger para generar documentación automática de la *API*. Esto no solo facilita la comprensión y el uso de la *API* por parte de otros desarrolladores, sino que también asegura que la documentación esté siempre actualizada con respecto a las últimas versiones de la *API*.

En cuanto al manejo de datos, aunque ya se ha discutido ampliamente el uso de Entity Framework Core, es importante mencionar que este *ORM* facilita enormemente las interacciones con la base de datos, minimizando la necesidad de escribir código SQL explícito y reduciendo el riesgo de errores.

El equipo ha configurado la autenticación mediante JWT (JSON Web Token) para asegurar las comunicaciones y el acceso a la *API*. Esto se realiza mediante la configuración de "JwtBearerDefaults" y la validación de los tokens JWT con parámetros que aseguran la autenticidad y validez de estos. Esta implementación asegura que las solicitudes a la *API* estén adecuadamente autenticadas, protegiendo así los recursos y datos sensibles del sistema.

El equipo ha seguido una filosofía de aprovechar al máximo las herramientas y bibliotecas proporcionadas por el ecosistema .NET, evitando la necesidad de desarrollar soluciones desde cero. Esta estrategia ha permitido centrarse en la lógica de negocio y en ofrecer un producto robusto y escalable, confiando en la solidez y fiabilidad de las soluciones estándar del *framework*.

5.4.2.2. Base de datos

Tanto la base de datos del negocio, como la base de datos de los resultados de los algoritmos de *machine learning*, son gestionadas por medio del *ORM* Entity Framework Core, el cual crea migraciones, y las actualiza en la base de datos. Jamás el equipo interactúa directamente con código del dialecto SQL de PostgreSQL.

EF *Core* viene a representar una especie de caja negra, por la cual el equipo desconoce y se despreocupa de la gestión entre la librería de EF *Core*, y la base de datos.

A continuación, se puede visualizar un diagrama (Ilustración 19) entidad relación de las tablas de la base de datos del negocio.

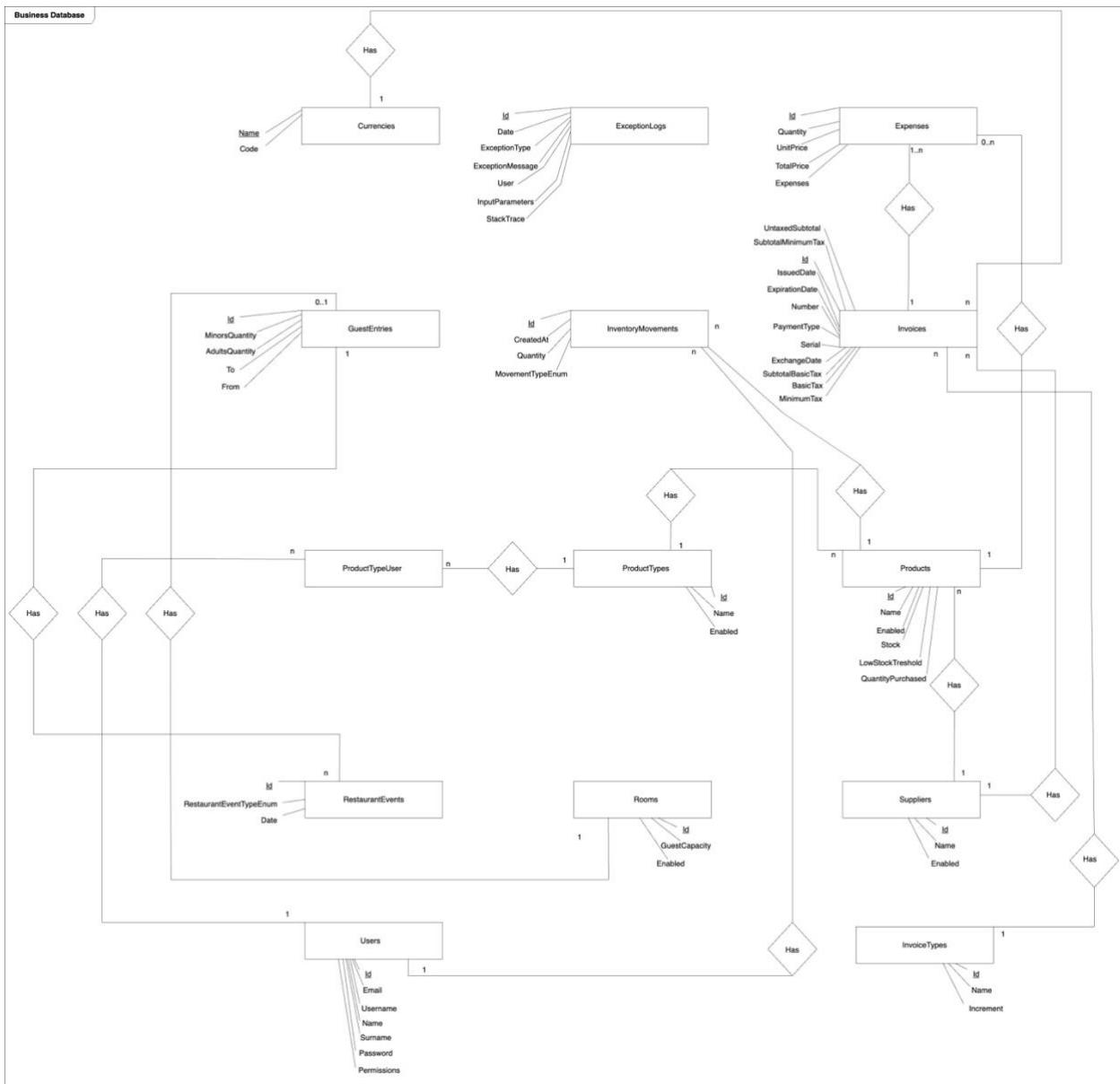


Ilustración 19 - Diagrama de modelo entidad relación de la base de datos del negocio

Cada una de las entidades/tablas de la base de datos tiene su significado.

5.4.2.2.1. Currencies

Currencies almacena las diferentes monedas que se utilizan en las transacciones con proveedores, siendo los pesos uruguayos y los dólares, las más comunes.

Los atributos incluyen un Code de tipo string que actúa como clave primaria y un Name, también de tipo string.

5.4.2.2.2. ExceptionLogs

Esta tabla es de uso interno del equipo de desarrollo. Permite identificar rápidamente las excepciones ocurridas cuando el producto es utilizado por el cliente. Se almacena información clave como: usuario, stack trace, tipo de excepción, y la llamada a la *API* (junto a los parámetros que causó el error).

Los atributos incluyen un Id, que es de tipo entero y es la clave primaria, Date, que es de tipo timestamp con zona horaria, ExceptionType y ExceptionMessage, ambos de tipo texto. El campo User es también de tipo texto, mientras que InputParameters y StackTrace son de tipo texto, permitiendo almacenar detalles sobre los parámetros de entrada y el rastreo de la pila de errores en la tabla "ExceptionLogs".

5.4.2.2.3. Expenses

Cada una de las filas de esta tabla se corresponde con una línea de cada factura. Es decir, cada expense es uno de los tantos gastos que puede haber en una factura.

Los atributos incluyen un Id como un entero que sirve como clave primaria. El InvoiceId y el ProductId también son de tipo entero, funcionando como claves foráneas que apuntan a las tablas de facturas y productos, respectivamente. Los campos Quantity, UnitPrice, y TotalPrice son de tipo double precision, representando la cantidad, el precio unitario, y el precio total de los gastos en la tabla "Expenses".

5.4.2.2.4. GuestEntries

Cada fila de esta tabla se refiere a uno o varios huéspedes asociados. Si por ejemplo una familia desea alojarse en el hotel, la misma representará una sola fila de esta tabla. De esta manera se puede saber en tiempo real cuántos huéspedes y habitaciones ocupadas hay en dicho momento.

Los atributos incluyen un Id, que es un entero y clave primaria, y RoomId, también de tipo entero, que es una clave foránea a la tabla de habitaciones. MinorsQuantity y AdultsQuantity son enteros que indican las cantidades de menores y adultos respectivamente. Los campos To y From son de tipo timestamp con zona horaria, y Enabled es de tipo booleano, indicando si la entrada está activa en la tabla "GuestEntries".

5.4.2.2.5. Inventory Movements

Esta tabla es la que altera el stock de los productos. Se refleja cada movimiento de stock de un producto, como, por ejemplo: consumos, vencimientos y roturas descienden el stock. Por otro lado, los ingresos (compras), aumentan el stock.

Los atributos incluyen un Id de tipo entero y actúa como clave primaria. Los campos UserId y ProductId son claves foráneas, de tipo entero, que se refieren a las tablas de usuarios y productos, respectivamente. CreatedAt es de tipo timestamp con zona horaria, mientras que Quantity es de tipo entero, y MovementTypeEnum también es de tipo entero, indicando el tipo de movimiento en la tabla "InventoryMovements".

5.4.2.2.6. Invoices

Esta tablata almacena las facturas que recibe el hotel, las cuales interesan únicamente aquellas relacionadas con proveedores de alimentos y bebidas. Cada factura está identificada por un Id, un número entero que actúa como clave primaria. El tipo de factura se define mediante el campo InvoiceType, también un número entero. La fecha de emisión de las mismas se registra en el campo IssuedDate, que es de tipo timestamp con zona horaria, proporcionando así la fecha y hora exacta en que fue generada la factura. El proveedor que emite la factura está identificado por el campo SupplierId, de tipo BigInt, y el número de factura queda almacenado en el campo Number como entero.

El método de pago utilizado se indica en el campo PaymentType, mientras que el número de serie/código asignado a la factura se guarda en el campo Serial, que es de tipo texto. Se incluye también el campo ExchangeRate para reflejar la tasa de cambio utilizada si la factura está en una moneda diferente a la moneda base del hotel. Aquellos montos que se relacionan con los impuestos se distribuyen en varios campos: SubtotalBasicTax, que almacena el subtotal sujeto a impuestos básicos. SubtotalMinimumTax, para el subtotal sujeto al impuesto mínimo. UntaxedSubtotal, que almacena el subtotal no gravado por ningún impuesto.

La moneda en la que se emite la factura se representa mediante el campo CurrencyCode, que almacena el código de la moneda, como USD o UYU (entre otros). Los valores de los impuestos aplicados a la factura se almacenan en los campos BasicTax y MinimumTax, ambos de tipo double precision, para reflejar con exactitud los montos correspondientes a los impuestos básicos y mínimos, respectivamente. También hay un atributo double llamado Discount, para almacenar el valor de descuento.

5.4.2.2.7. ProductTypeUser

Esta tabla almacena en cada uno de sus registros, una referencia a un usuario, y una referencia a un tipo de producto. ¿Para qué? En el caso que haya bajo stock de un producto de este tipo de producto (stock menor al atributo "LowStockTreshold" de la tabla "Products"), se les envía un mail a los usuarios suscritos a este tipo de alertas.

Los atributos incluyen un LowStockAlertUserId y un ProductTypeId, ambos de tipo entero, con claves foráneas que hacen referencia a los usuarios que reciben alertas de bajo stock y los tipos de producto, respectivamente, en la tabla "ProductTypeUser".

5.4.2.2.8. ProductTypes

Esta tabla agrupa las diferentes categorías de los productos.

Los atributos incluyen un Id de tipo entero como clave primaria, un Name de tipo texto que describe el nombre del tipo de producto, y un campo Enabled de tipo booleano que indica si el tipo de producto está activo en la tabla "ProductTypes".

5.4.2.2.9. Products

Todos los productos que existen en el hotel (en principio alimentos y bebidas), se almacenan en esta tabla.

Los atributos incluyen un Id de tipo entero como clave primaria. El campo ProductTypeId es una clave foránea que referencia a la tabla de tipos de productos, y SupplierId es también una clave foránea que apunta a la tabla de proveedores. Los campos Stock y LowStockThreshold son de tipo double precision, representando el stock actual y el umbral de alerta de bajo stock en la tabla "Products". El campo textual Name, describe al producto. Luego tenemos al campo double precision llamado QuantityPurchased, el cual refleja la cantidad de unidades compradas de dicho producto.

5.4.2.2.10. RestaurantEvents

Almacena un registro por cada vez que ingresa un huésped o grupo de huéspedes, al comedor, ya sea para desayunar, almorzar, merendar o cenar.

Los atributos incluyen un Id de tipo entero y sirve como clave primaria. GuestEntryId es una clave foránea que apunta a la tabla de entradas de huéspedes. RestaurantEventTypeEnum es de tipo entero y se utiliza para identificar el tipo de evento de restaurante, mientras que Date es de tipo timestamp con zona horaria en la tabla "RestaurantEvents".

5.4.2.2.11. Rooms

Refiere a las habitaciones donde se hospedan los huéspedes durante su estadía.

Los atributos incluyen un Id de tipo entero como clave primaria, un GuestCapacity de tipo entero que indica la capacidad de la habitación en cuanto a número de huéspedes, y un campo Enabled de tipo booleano que señala si la habitación está activa o no en la tabla "Rooms".

5.4.2.2.12. Suppliers

Representa a los proveedores de productos del hotel.

Los atributos incluyen un Id de tipo bigint como clave primaria (RUT), un Name de tipo texto que describe el nombre del proveedor, y un campo Enabled de tipo booleano que indica si el proveedor está activo en la tabla "Suppliers".

5.4.2.2.13. Users

Los diferentes tipos de usuarios del sistema se almacenan aquí (ya sean usuarios *web* o de la *app móvil*).

Los atributos incluyen un Id de tipo entero como clave primaria, un Email de tipo texto que almacena la dirección de correo electrónico del usuario, un Username de tipo texto que guarda el nombre de usuario, así como los campos Name y Surname, ambos de tipo texto, que almacenan el nombre y apellido del usuario, respectivamente. Además, el campo Password es de tipo texto y guarda la contraseña cifrada del usuario. Finalmente, el campo Permissions es de tipo texto y almacena los permisos asignados al usuario en la tabla "Users".

5.4.2.2.14. InvoiceTypes

Esta tabla almacena los diferentes tipos de factura existentes, por medio de datos como: id (número), name (texto) e increment (bool).

5.4.2.2.15. __EFMigrationsHistory

Esta tabla contiene la información de las migraciones de la base de datos creada por el *ORM EntityFramework Core*.

5.4.2.3. Machine Learning

El equipo determinó que el entrenamiento para el módulo de *machine learning*, va a constar de la ejecución periódica de una triggered function de Azure *DevOps*. Esto significa que cada cierto período de tiempo (por ejemplo, una semana), el algoritmo de entrenamiento se ejecutaría (gracias a los datos históricos provistos por la base de datos de negocio), para luego ejecutar el modelo y producir los outputs esperados.

Esta función automatizaría el proceso de entrenamiento y evaluación de modelos de *machine learning*, permitiendo una actualización continua y mejorada de los algoritmos de predicción, sin necesidad de que se tenga que ejecutar de manera manual.

5.4.2.3.1. Librerías utilizadas

Como es de previo conocimiento, se utilizará el lenguaje Python, en complemento con librerías conocidas para data frames y *machine learning*. Las más importantes son:

- scikit-learn (árboles de decisión)

- pandas (para manipular y analizar los datos)
- numpy (para realizar cálculos numéricos)

5.4.2.3.2. Modelos

La triggered function entrenará algoritmos/modelos para realizar predicciones acerca del principal tema de preocupación del cliente temas relevantes del negocio para el cliente, como, por ejemplo:

- Modelo 1: Diseñado para obtener los montos referidos a los gastos totales para un período de tiempo en particular.
- Modelo 2: Diseñado para determinar los números referidos al stock de productos (alimentos y bebidas) que se necesita tener como mínimo para poder satisfacer la demanda de los clientes en un período de tiempo en particular.

Para ambos modelos mencionados anteriormente, se utilizarán modelos de regresión porque están diseñados específicamente para predecir valores numéricos continuos, lo que es esencial cuando se necesita estimar con precisión un valor específico, como precios o cantidades. Por otro lado, no se utilizarán árboles de decisión, ya que existe una tendencia de estos a sobre ajustarse a los datos de entrenamiento y pueden no gestionar bien relaciones complejas, lo que podría resultar en un modelo de predicción menos preciso que el elaborado con regresión lineal.

5.4.2.3.3. Datos de input

El input del modelo 1 consta de las facturas almacenadas, osea, los registros de la tabla "Invoices" que anteriormente fue descrita.

El input del modelo 2 consta de: 1) Los movimientos de inventario, específicamente, los registros de la tabla "InventoryMovements" que también anteriormente fue descrita. 2) Datos de los huéspedes (ya que el sistema también deja registro de cada hospedaje, junto a la cantidad de huéspedes, y las fechas de ingreso y egreso). De esta manera se puede establecer tendencias entre los consumos y uso de productos (movimientos de inventario) respecto a la cantidad de huéspedes en un determinado momento.

Luego del entrenamiento de los modelos, se procede a ejecutar dichos algoritmos con datos reales como input. En esta fase, el modelo entrenado se aplica para generar predicciones o insights que puedan ser utilizados para la toma de decisiones. Los resultados generados son almacenados en la base de datos PostgreSQL anteriormente mencionada.

5.4.2.3.4. Datos de output

Para el modelo uno, el output consta de la agregación de una nueva tupla en la tabla "TotalExpenses" de la base de datos exclusiva de los datos de *machine learning*. Dicha tabla va a

tener los datos: "Id" (número autoincrementable), "Amount" (variable numérica decimal, expresada en pesos uruguayos), "From" (fecha desde) y "To" (fecha hasta).

Por otro lado, en el modelo 2, el output no constará de una sola nueva tupla, sino que habrá tantas tuplas como productos existentes, debido a que se realiza una predicción por cada producto. Cada una de estas tuplas irán almacenadas en la tabla "MinimumRequiredStock", con los datos: "Id" (número autoincrementable), "ProductName" (string), "Stock" (variable numérica decimal, ya que el stock puede ser un entero, o decimal a causa de los productos pesables), "From" (fecha desde) y "To" (fecha hasta).

Las tablas "MinimumRequiredStock" y "TotalExpenses", no fueron diagramadas en un modelo entidad relación debido a que como se ha comentado, el módulo de *machine learning* no es la prioridad ahora mismo.

5.5. Seguridad del sistema

La autenticación es un componente crítico para garantizar la seguridad en *ChefCheck-Manager*. El sistema utiliza JWT (JSON Web Tokens) para gestionar el proceso de autenticación de los usuarios. Cuando un usuario inicia sesión, sus credenciales se verifican y se genera un token JWT que contiene la información del usuario y sus permisos. Este token tiene un tiempo de expiración predefinido y es enviado con cada solicitud al *backend* .NET, donde es validado para asegurar la legitimidad del acceso y los permisos correspondientes.

En *ChefCheck-Manager* existen varios roles, cada uno con diferentes niveles de acceso y permisos, lo que garantiza que los usuarios solo puedan interactuar con las funcionalidades autorizadas para su rol específico. Además, se emplea autenticación de base de datos con usuario y contraseña para proteger el acceso a los datos sensibles.

5.5.1. Seguridad en el Transporte de la Información

El sistema utilizará HTTPS para garantizar que toda la información transmitida entre los *frontends* (Flutter y Angular) y la *API* .NET esté cifrada y protegida contra posibles interceptaciones. El uso de HTTPS asegurará la confidencialidad de los datos, proporcionando una capa adicional de seguridad que minimiza los riesgos de ataques como el man in the middle.

5.6. Deployment en Azure

A continuación, se describe el proceso que realizará el equipo para hacer el *deployment* del sistema bajo la infraestructura de Azure, el cual será realizado luego de la entrega administrativa en el portal de gestión.

5.6.1. Base de datos PostgreSQL

En principio, este proceso será aplicado para la base de datos de negocio, pero posteriormente, con la implementación del módulo de *machine learning*, se repetirán los pasos que sean considerados como necesarios.

5.6.1.1. Acceso al portal de Azure

Para desplegar la base de datos PostgreSQL del proyecto en Azure, primeramente, se debe ingresar con las credenciales correspondientes de la cuenta de Azure del Hotel. Una vez dentro, se accede a la opción "Crear un recurso" ubicada en la parte superior izquierda del portal. Utilizando el buscador, se escribe "Azure Database for PostgreSQL" y se selecciona la opción "Azure Database for PostgreSQL Flexible Server" para iniciar la configuración del servidor [1].

5.6.1.2. Configuración del servidor PostgreSQL

El equipo va a contar con una suscripción de producción previamente configurada, lo que permite desplegar los recursos directamente en ella. Una vez en la pantalla de configuración del servidor PostgreSQL, se selecciona esta suscripción para garantizar que todos los recursos estén asociados al entorno de producción de *ChefCheck-Manager*. Luego, el equipo procede a elegir el grupo de recursos, seleccionando el ya creado *ChefCheck-ManagerResources*, que centraliza todos los recursos del proyecto.

A continuación, es necesario asignar un nombre único al servidor de la base de datos PostgreSQL. El equipo va a ingresar por un nombre como "chefcheckdb", lo que automáticamente generará una URL del servidor (la cual es utilizada en el connection string, dentro de la *API*). Dado que el hotel opera en Uruguay, se seleccionará la región *South Brazil* para optimizar la latencia y mejorar el rendimiento del sistema, asegurando que el servidor de base de datos esté lo más cercano posible a los usuarios y al equipo de trabajo.

5.6.1.3. Definición de las credenciales y configuración de versión de la base de datos

El equipo asignará un nombre de usuario para el administrador del servidor, eligiendo una contraseña que cumpla con las políticas de seguridad de Azure. Posteriormente, se seleccionará la versión más reciente de PostgreSQL para garantizar la compatibilidad con los requisitos del proyecto *ChefCheck-Manager*. También se seleccionará el poder cómputo y el almacenamiento de acuerdo con las necesidades actuales del proyecto, optando por una configuración estándar, como 4 vCores y 512 GB de almacenamiento (la cual será de utilidad, al menos en los primeros meses de vida del sistema en producción).

5.6.1.4. Configuración de red

El equipo configurará el acceso a la red para la base de datos PostgreSQL utilizando acceso público. Mientras la *API* del proyecto aún no está publicada, se mantendrá el acceso público para facilitar la conexión de los desarrolladores y permitir el trabajo. Esto permite un acceso más amplio durante la fase de desarrollo.

Una vez que la *API* se publique y esté en funcionamiento en Azure App Service, se configurará el acceso restringido, añadiendo la dirección IP del servicio donde esté alojada la *API*. Este cambio garantizará un acceso más seguro, limitando la conexión a la base de datos exclusivamente a la *API*, asegurando así una integración controlada y segura. Mientras tanto, el acceso público es una solución temporal para el equipo de desarrollo. En el anexo [Azure App Service](#) se puede encontrar información acerca de lo que es Azure App Service.

5.6.1.5. Revisión y creación del servidor

Tras revisar todos los detalles configurados, el equipo procederá a clickear en el botón "*Create*". El proceso de *deployment* demora unos minutos mientras se setea todo. Una vez finalizado, el equipo puede acceder a la página de información general del servidor para obtener los detalles de conexión, como el nombre del servidor y las credenciales del usuario administrador. Esto permitirá al equipo conectarse al servidor PostgreSQL y comenzar a gestionar los datos del proyecto utilizando el cliente PostgreSQL correspondiente (así como ya actualizar el *connection string* de la base de datos, dentro de la *API*).

5.6.2. API .NET

5.6.2.1. Desplegar la API .NET mediante Azure App Service

Para publicar una *API* en ASP.NET *Core* usando Azure App Service, se utilizará un *pipeline* en Azure *DevOps* que le automatiza al equipo, el proceso de construir el ejecutable y el despliegue. Este *pipeline* optimiza el flujo de trabajo y garantiza que cada cambio en el código pase por un proceso controlado antes de llegar a producción.

5.6.2.2. Uso de un despliegue que no es *self-contained*

En el caso del equipo, la *API* no es *self-contained*, lo que significa que no se incluyen todas las dependencias en el paquete de despliegue (el ejecutable de la *API* no cuenta con todas las dependencias de .NET necesarias para su utilización). En lugar de eso, se aprovechará el *runtime* de .NET *Core* que ya está disponible en Azure App Service. Esto reduce el tamaño del despliegue y hace más eficiente el proceso al aprovechar la infraestructura que Azure ya ofrece.

5.6.2.3. Configuración del *pipeline* en Azure DevOps

El primer paso en la configuración del *pipeline* es agregar una *task* que asegure el uso de una versión específica del *SDK* de .NET Core. Esto garantiza que todas las fases del *pipeline*, como restaurar, compilar, probar y publicar, utilicen siempre la misma versión del *SDK* que está disponible en Azure App Service. Este paso se configura seleccionando la tarea Use .NET Core en el *pipeline* y especificando la versión que se necesita. Este paso debe ser el primero para que todas las fases posteriores trabajen con esa versión del *SDK*.

5.6.2.4. Automatización de las pruebas y despliegue

El *pipeline* de Azure DevOps también puede incluir una fase de pruebas antes del despliegue, lo que garantiza que solo el código que pase todas las pruebas automatizadas llegue a producción. Esta automatización asegura que la *API* no contenga errores en los *tests* antes de ser desplegada.

Esto es algo que el equipo consideró, ya que aporta mayor seguridad. Únicamente se puede publicar una versión, la cual todos sus *tests* aprueben.

5.6.2.5. Despliegue en el mismo grupo de recursos

La *API* y otros recursos relacionados, como la base de datos PostgreSQL de *ChefCheck-Manager*, se encontrarán en el mismo grupo de recursos previamente creado en Azure, llamado "*ChefCheck-ManagerResources*". Esto facilitará la administración conjunta de todos los recursos de la aplicación, optimizando tanto la gestión como la escalabilidad. Tener todos los componentes en un mismo grupo permite aplicar políticas y administrar el ciclo de vida de manera centralizada.

5.6.3. Web App Angular

5.6.3.1. Despliegue de una aplicación Angular en Azure Static Web Apps

Para desplegar una aplicación Angular en Azure Static Web Apps (más información de este servicio en el anexo [Azure Static Web](#)), el equipo seguirá un proceso sencillo a través del portal de Azure. Utilizando el mismo grupo de recursos que será previamente creado para el proyecto *ChefCheck-Manager*, se configurará el despliegue de la aplicación estática en Azure y se vinculará el código fuente almacenado en el repositorio de GitHub.

5.6.3.2. Configuración inicial

El primer paso consistirá en acceder al portal de Azure y seleccionar la opción "Crear un recurso". En el buscador del portal se ingresará "Azure Static Web Apps" y se seleccionará esta opción para iniciar el proceso de configuración de la aplicación estática. En la sección de configuración básica, se utilizará la suscripción de Azure correspondiente y se seleccionará el grupo de recursos que ya

existirá para este entonces, "*ChefCheck-ManagerResources*", donde se desplegarán todos los recursos relacionados con la aplicación.

Luego, se asignará un nombre descriptivo a la aplicación estática, como "*chefcheck-Angular-app*", y se seleccionará el plan de *hosting Free* (adecuado para proyectos personales o de prueba). El equipo eligió GitHub como fuente del código y vinculará el repositorio donde estaba alojada la aplicación Angular. Para ello, será necesario iniciar sesión con una cuenta de GitHub y especificar el repositorio y rama que se utilizarían para el despliegue.

5.6.3.3. Configuración de la aplicación Angular

En la sección de detalles de la compilación, se seleccionará Angular en el menú de *Build Presets*, lo que permitirá que Azure Static Web Apps configure automáticamente los detalles necesarios para construir la aplicación. La ubicación del código de la aplicación será especificada como *dist/Angular-basic*, lo cual indicará dónde encontrar el código ya compilado de la aplicación Angular. Luego se procederá a crear la aplicación.

5.6.3.4. Vinculación del dominio propio

El equipo del Hotel Tío Tom comprará un dominio en la *web* de GoDaddy, el cual se vinculará a la aplicación desplegada en Azure. Una vez que la aplicación Angular se encuentre desplegada y accesible mediante la URL predeterminada proporcionada por Azure, el equipo procederá a configurar el dominio personalizado.

Desde la sección de *Custom domains* en el portal de Azure, se añadirá el dominio adquirido. El equipo gestionará las configuraciones de DNS desde GoDaddy, apuntando el dominio al servicio de Azure mediante un registro CNAME, asegurándose de que el tráfico hacia el dominio nuevo sea redirigido correctamente a la aplicación estática en Azure.

5.6.3.5. Visualización de la aplicación

Una vez que se complete la configuración del dominio y las compilaciones de la aplicación sea exitosa, el equipo podrá acceder a la aplicación Angular a través del dominio personalizado proporcionado por GoDaddy. Azure se encargará del certificado SSL automáticamente, asegurando que la aplicación se encuentre disponible a través del protocolo HTTPS.

La integración con GitHub también permitirá que cada cambio realizado en el repositorio desencadene automáticamente una nueva compilación y despliegue en Azure Static Web Apps, lo que garantizará que la aplicación siempre esté actualizada con la última versión del código.

5.6.4. Flutter Google PlayStore App

5.6.4.1. Despliegue de la *app* en Google Play Store

Para que el equipo pueda completar con éxito el despliegue inicial de la aplicación móvil del hotel en la Google Play Store, utilizará una cuenta de desarrollador de Google ya creada por el hotel. La aplicación inicialmente será publicada en modo de *Closed Testing*, lo que significa que está disponible solo para un grupo selecto de usuarios dentro de la organización del hotel, permitiendo realizar pruebas internas antes de avanzar a un despliegue público.

5.6.4.2. Publicación en la Google Play Console

El equipo utilizará la *web* de la Google Play Console para gestionar el proceso de publicación de la *app*. Una vez creada la aplicación dentro de la consola, se configurarán los detalles clave, como el nombre de la aplicación, la descripción, los iconos, capturas de pantalla y la categoría de la aplicación. Tras esta configuración, se generará un archivo .aab (*Android App Bundle*) que será subido a la consola para su revisión y posterior despliegue en el canal de *Closed Testing*.

En esta fase, la aplicación será habilitada únicamente para usuarios autorizados dentro del hotel, lo que permitió realizar pruebas internas y garantizar que todas las funcionalidades estuvieran en orden. El equipo asignó un grupo de *testers* dentro de la propia organización del Hotel Tío Tom, quienes utilizarán la aplicación en un entorno controlado y proporcionarán retroalimentación crucial para futuras mejoras.

5.6.4.3. Distribución privada de la *app* para la organización

La aplicación será publicada como una aplicación privada para la organización del hotel cuando pase a producción en diciembre. Esto significa que solo los empleados y miembros autorizados podrán acceder y descargarla mediante las herramientas de administración de dispositivos móviles del hotel.

Utilizando la Google Play Console, en el futuro se configurará la aplicación para que esté disponible exclusivamente en la cuenta de la organización del hotel. Esto se logrará mediante el uso de la opción de Managed Google Play en la consola, lo cual permitirá que las aplicaciones empresariales se distribuyan de manera controlada únicamente dentro de una organización.

5.6.4.4. Costos del despliegue en la modalidad *Closed Testing*

El despliegue de la aplicación en modo *Closed Testing* a través de la Google Play Store no tiene costos adicionales más allá del costo único de la cuenta de desarrollador de Google, que tiene un precio de 25 USD. Este costo es un pago único que permite publicar aplicaciones en la Play Store de manera indefinida, ya sea para pruebas internas o despliegue público en el futuro (*release*).

5.6.5. Azure Triggered Function

Se utilizará el servicio de Azure Functions.

El entorno de desarrollo Rider tiene un tipo de proyecto específico para este tipo de funciones (Ilustración 20).

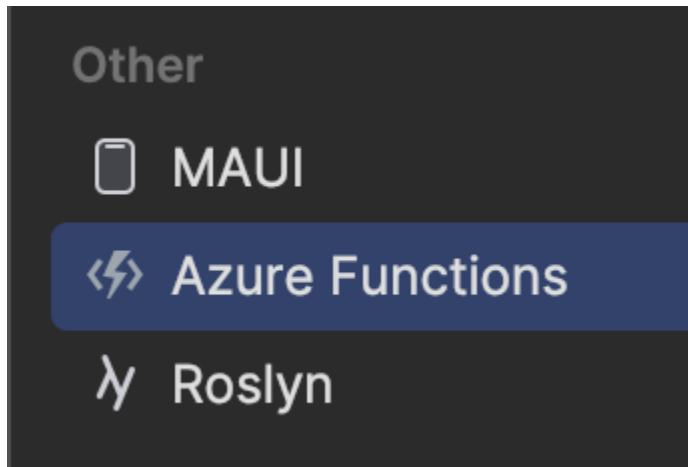


Ilustración 20 - Azure Functions en Rider

Por medio de comandos, Rider permite que la creación, codificación y prueba de la función pueda realizarse localmente.

El comando "func start" se utiliza para ejecutar el servidor local y probar la función.

Una vez finalizada la función, se publica en azure con este comando "func azure functionapp publish <functionName>".

En el anexo [Azure Functions](#), se puede encontrar más contexto sobre lo que es Azure Functions.

5.7. Evolución de los requerimientos no funcionales

En el anexo [Evolución de los requerimientos no funcionales](#), se visualiza el desglose de requerimiento por requerimiento, en cuanto a su estado y/o cumplimiento al momento de la entrega de esta documentación. Se puede volver a consultar los requerimientos no funcionales en esta sección [Requerimientos no funcionales](#).

6. Gestión del proyecto

6.1. Herramientas para la gestión

6.1.1. JIRA

JIRA se utiliza como herramienta principal para la gestión de tareas, seguimiento del progreso, y facilitamiento de las reuniones y retrospectivas.

Esta aplicación permite organizar el trabajo a través de diferentes tipos de incidencias. Las mismas son: Épicas, *user stories*, tareas y *bugs*.

Las épicas ofrecen menos detalle, ya que contiene un título que se corresponde a información a muy alto nivel.

Cada *user story* se ubica dentro de una épica. Por otro lado, los criterios seleccionados por el equipo para el definition of done, y definition of ready, se encuentran en la sección de [Gestión de Calidad](#).

Dentro de cada *user story* se ubican las tareas. El equipo decidió que la tarea sea la unidad de menor granularidad del proyecto, es decir, la pieza con mayor nivel de información. El equipo determinó que las tareas deben ser lo suficientemente pequeñas como para que no amerite que dos integrantes del equipo deban relaizarlas, sino uno solo.

El equipo consideró que JIRA es ideal para la gestión de proyectos debido a la capacidad de organizar el trabajo a través de tableros interactivos y diversos tipos de incidencias. Se utilizan los informes de *burnup* y *burndown Chart* para visualizar el progreso, facilitando la documentación de cada *sprint*. Además, el tablero permite asignar tareas, establecer prioridades, añadir comentarios y archivos adjuntos, y realizar un seguimiento del tiempo dedicado a cada tarea, lo que mejora la colaboración y eficiencia del equipo.

Se eligió JIRA en lugar de Azure *DevOps* debido a su uso generalizado entre los miembros del equipo, facilitando la adopción y reduciendo el tiempo de capacitación.

6.2. Etapa de investigación

Anteriormente el equipo comentó acerca de la primera etapa del proyecto, la etapa de investigación. Esta etapa, tiene un comienzo simbólico el 30 de enero, debido a que fue la jornada que brindó la ORT, para comunicar a los alumnos información clave sobre los proyectos de fin de carrera. Desde ese día, en el cual el equipo se formó, se da comenzada la etapa de investigación, la cual consta de diferentes etapas descritas a continuación.

6.2.1. Definición de los requerimientos

El primer paso en la fase de investigación fue la definición de los requerimientos del *software*. A través de la visita al hotel, se identificaron las necesidades y objetivos clave del proyecto. Estas reuniones fueron esenciales para garantizar que todos los aspectos críticos del sistema fueran considerados desde un principio, ya que los requerimientos son estáticos. Lo que también cuenta como parte definición de requerimientos, es el trabajo previo de Francisco Cabanillas con el hotel. Si bien hace años desarrolló una versión muy reducida respecto a lo que será el sistema luego de la entrega de este proyecto, el ya venía recopilando información acerca de los requerimientos nuevos para este proyecto.

6.2.2. Investigación de APIs y tecnologías

Un enfoque principal de esta etapa fue la investigación de las *APIs* de la empresa de facturación electrónica FacturaLista y DGI, ya que estas integraciones son cruciales para el correcto funcionamiento del *software*. Se llevó a cabo un análisis exhaustivo de la documentación técnica y las capacidades de estas *APIs* para asegurar una integración exitosa. Además, se evaluaron diferentes tecnologías y herramientas que podrían utilizarse en el proyecto, considerando factores como la escalabilidad, compatibilidad y facilidad de uso.

6.2.3. Comienzo de la documentación

Paralelamente a la investigación, se inició la documentación del proyecto. Este proceso incluyó la creación de documentos que describen los requerimientos, las decisiones tomadas sobre las tecnologías a utilizar y las especificaciones técnicas necesarias para la integración con las *APIs* de terceros. Esta documentación fue crucial para mantener una referencia clara y accesible durante todo el desarrollo.

6.2.4. Definición del ciclo de vida y metodología de trabajo

Durante esta etapa, también se definió el ciclo de vida del proyecto y la metodología de trabajo a seguir. Se optó por un enfoque basado en Scrum, adaptado a las necesidades específicas del proyecto. Esta decisión permitió una organización estructurada del trabajo, con iteraciones cortas de dos semanas, lo que facilitó la adaptación a cambios y la entrega continua de valor.

6.2.5. Esfuerzo dedicado

A continuación, se presenta un resumen (Ilustración 21) del esfuerzo dedicado a cada una de las áreas clave durante esta etapa de investigación. La etapa transcurre desde el 30 de enero de 2024 hasta el 17 de junio de 2024 (fecha de comienzo del primer *sprint*). El *sprint* 0 se encuentra contenido dentro de la etapa de investigación. Recordemos que este *sprint* comienza oficialmente el día que se aprobó el proyecto (22 de abril de 2024).

Tema	Horas	Descripción
(1) - Selección de herramientas y tecnología	7	Investigación, comparación de pro y contras de cada una y selección
(2) - Reuniones	20	Reuniones entre el equipo (<i>Scrum daily, planning, review, retrospective</i>) y con la tutora.
(3) - Definición de los requerimientos del <i>software</i>	32	Captura y especificación de los requerimientos (incluye reunión presencial en el hotel del cliente a fines de marzo de 2024)
(4) - Investigación <i>APIs</i> de terceros	13	Investigación de la propuesta de FacturaLista, y cómo sería su integración respecto a la <i>API</i> de DGI.
(5) - Documentación	50	Horas dedicadas a documentación.

Ilustración 21 - Resumen de dedicación

La etapa de investigación (que duró 122 horas aproximadamente), fue crucial para establecer una base sólida para el desarrollo del *software*, garantizando que todas las decisiones estuvieran fundamentadas en un análisis exhaustivo y que el equipo estuviera alineado en cuanto a los objetivos y el enfoque del proyecto.

6.3. Etapa de desarrollo

6.3.1. Scrum

Como se mencionó previamente, el equipo optó por el uso de Scrum como metodología para la fase de desarrollo del *software*, adaptando este marco de trabajo a las necesidades específicas del proyecto. Esta decisión fue justificada y explicada en detalle en secciones anteriores del documento. En los siguientes párrafos, se explicará más a fondo cómo se implementó Scrum, incluyendo una descripción del equipo de Scrum, los artefactos utilizados y los eventos clave del proceso.

La implementación de Scrum no fue una adopción directa del método estándar; en cambio, se ajustó para alinear mejor con las dinámicas y restricciones del equipo. Este capítulo detalla cómo estos ajustes permitieron que el equipo pueda gestionar eficazmente el desarrollo del *software*, maximizando la productividad. Se comentará acerca de los roles del equipo, los artefactos que

guían el flujo de trabajo y los eventos que estructuran cada *sprint*, proporcionando una visión de la operativa diaria bajo este marco ágil adaptado.

6.3.1.1. Roles Scrum

Product owner

- Integrante: Maximiliano Correa (gerente operacional del Hotel Tío Tom)
- Responsabilidades:
 - Definir los requisitos del producto y priorizar el product *backlog*.
 - Asegurar que el equipo entienda los ítems del product *backlog* al nivel necesario.
 - Maximizar el valor del producto resultante del trabajo del equipo de desarrollo.

Scrum master

- Integrante: Martín Sosa
- Responsabilidades:
 - Facilitar los eventos de Scrum (*sprint planning*, *daily scrum*, *sprint review*, *sprint retrospective*).
 - Asegurar que se sigan las prácticas y principios de Scrum.
 - Ayudar al equipo eliminando impedimentos.
 - Actuar como un facilitador dentro del equipo.

Developers

- Integrantes: Francisco Cabanillas, Andrés Quintero y Martín Sosa.
- Responsabilidades:
 - Realizar el trabajo de desarrollo del *software*.
 - Crear incrementos del producto *sprint* a *sprint*.
 - Asegurar la calidad del *software* a través de pruebas y mantenimiento de la definition of done

6.3.1.2. Artefactos de Scrum

Product backlog

Es la lista ordenada de todo lo necesario para mejorar el producto, gestionada por el Scrum *master*, bajo los convenios acordados con el *product owner*.

Sprint backlog

Es un conjunto de ítems seleccionados del product *backlog* para desarrollar durante el *sprint*, incluyendo un plan para el *sprint*.

Incremento

La suma de todos los ítems del *backlog* completados durante un *sprint* y los anteriores *sprints*, que cumplen con la *definition of done*.

6.3.1.3. Eventos de Scrum

6.3.1.3.1. Sprint planning

Este evento marca el comienzo de cada *sprint* y es crucial para establecer el trabajo para las próximas dos semanas (duración del *sprint* establecida). En esta reunión, participan el *product owner*, el Scrum *master*, y todos los *developers*. El *product owner* presenta los ítems prioritarios del product *backlog*, y el equipo selecciona aquellos que se compromete a completar durante el *sprint*, formando así el *sprint backlog*.

Como en la adaptación de Scrum del equipo el *product owner* no participa de esta instancia, el mismo es sustituido por Francisco Cabanillas, quien es el *project manager*, y además cuenta con todo el conocimiento de negocio gracias a que trabajó algunas temporadas en el Hotel, junto a Maximiliano Correa, el *product owner*. Esto fue algo que tanto Maximiliano Correa, como la dirección del hotel, lo autorizaron.

Se discuten los requisitos, se negocian los alcances, y se realiza una estimación de los ítems a trabajar, utilizando una técnica personalizada del equipo (la cual será detallada más adelante), para llegar a un consenso sobre el esfuerzo requerido. Esta planificación toma en cuenta la capacidad del equipo y las posibles variaciones en la disponibilidad de los miembros debido a compromisos externos.

Todos los *sprints* del proyecto tuvieron una duración de dos semanas, menos el *sprint* 9 (7/10/2024 - 17/10/2024), el cual duró 10 días. Esto se debe a que se acercaba la fecha de entrega en gestión, y el equipo quería culminar el desarrollo del sistema en dicho momento, para que los tres integrantes del equipo puedan dedicarse al 100 por ciento en lo que corresponde a documentación.

6.3.1.3.2. Daily Scrum

Dado que las reuniones diarias no son posibles, el equipo ha decidido reunirse una vez a la semana (jueves). El motivo de haber elegido el día jueves, es porque es un día que se encuentra casi a la mitad de la semana de trabajo, por lo cual, debería haber avances (ya que los *sprints* siempre comienzan los lunes). Estas reuniones son breves y centradas en la actualización del progreso hacia el objetivo del *sprint*. Cada miembro del equipo comparte avances, planifica sus actividades para el día y discute impedimentos que podrían necesitar atención del Scrum *master*. Esta adaptación permite mantener una comunicación fluida y continua, asegurando que todos los miembros estén alineados y puedan colaborar eficazmente en los objetivos del *sprint*.

Hubo ocasiones donde el equipo se reunió más veces. Eso se fue decidiendo *sprint* a *sprint* dependiendo de la complejidad y las necesidades de este. En la adaptación Scrum del equipo, se utiliza el término *daily*, pese a que lo más lógico sería decir *weekly*, ya que por lo general hubo una sola reunión semanal, pero para hacer hincapié en que es un evento de Scrum, se sigue utilizando la palabra *daily*.

6.3.1.3.3. Sprint review

Al final de cada *sprint*, se realiza el *sprint review*, donde el equipo espera obtener un incremento. Este evento es una oportunidad para demostrar el trabajo completado y discutir sobre el product *backlog* en el contexto de lo que se ha entregado. El *product owner* tiene un rol muy importante aquí, obteniendo *feedback* que puede influir en la priorización de futuros ítems del *backlog*. Este *feedback* es esencial para adaptar el producto a las necesidades reales del cliente y el mercado. Además, se evalúa el cumplimiento del trabajo realizado, con respecto a los objetivos del *sprint*, identificando oportunidades de mejora y éxito.

6.3.1.3.4. Sprint retrospective para la mejora continua

Este evento se centra en la mejora del equipo y en la optimización de sus procesos. Se lleva a cabo después del *sprint review* y antes del próximo *sprint planning*. El equipo reflexiona sobre el *sprint* pasado, para identificar qué se hizo bien, qué se puede mejorar, y cómo implementar mejoras. Se incentiva a que haya una discusión abierta y honesta, que permita al equipo comprometerse a mejorar para los próximos *sprints*.

La metodología aplicada incluye la identificación de los aspectos positivos y negativos del *sprint*, la generación de ideas para abordar los desafíos y la planificación de implementaciones que promuevan la eficiencia y la efectividad del equipo.

A través de las retrospectivas, el equipo identifica oportunidades de mejora en sus procesos y rendimiento, asegurando un desarrollo continuo y eficiente del producto y del entorno de trabajo.

El equipo utilizó un board en Miró para plasmar la retrospectiva. La idea fue que cada integrante del equipo agregue notas dentro de las cuatro categorías a evaluar. Luego de eso se discutirán los resultados. El detalle de cada sesión de retrospectiva se encuentra al final del análisis de cada *sprints*, en los anexos que van desde el [Documentación del Sprint #1](#) hasta el [Documentación del Sprint #9](#).

6.3.1.4. Estimación del esfuerzo

Compromiso respecto al tiempo

La facultad recomienda que cada integrante del equipo dedique, en promedio, 14 horas semanales al proyecto. Para una gestión eficaz del tiempo y los recursos, se ha establecido que cada *story*

point equivalga a una hora de trabajo. Esta equivalencia proporciona una representación clara y precisa del esfuerzo necesario, facilitando así la planificación detallada y el seguimiento del progreso del proyecto.

Técnica de estimación utilizada

En las reuniones de *sprint planning*, se utiliza una técnica personalizada del equipo para la estimación del esfuerzo necesario para cada *user story* incluida en el *sprint backlog*. Cada miembro del equipo contribuye con su estimación del esfuerzo, aplicando la equivalencia establecida de que un *story point* corresponde a una hora hombre (Ilustración 22/Ilustración 22).

Mediante un proceso iterativo de discusión y votación, se alcanza un consenso sobre los *story points* asignados a cada *user story*. Este método asegura que se comprenda más detalladamente acerca de las expectativas de la carga laboral, lo cual es vital para la organización de cada *sprint*.

Story point estimate

1

Ilustración 22 - Atributo "Story point estimate" donde se lleva el registro del estimado en Jira para cada la actividad

Estimando esfuerzo en JIRA

JIRA cuenta con un mecanismo para trackear los *story points* de cada *user story*. A su vez, nos informa del porcentaje de completitud de estas, para tener una idea del esfuerzo en tiempo real.

JIRA es extensible. Cada *user story*, o cualquier otro tipo de incidencia (*bugs, tasks, etc*), cuentan con una serie de atributos, como, por ejemplo: título, descripción, etc. Pero al ser extensible, permite que se creen nuevos atributos.

Uno de los nuevos atributos creados, el equipo lo llamó "Total Effort" (Ilustración 23/Ilustración 23). Es de tipo numérico, por lo cual, cuando un *developer* termina su desarrollo, debe ingresar la cantidad de *story points* que la misma le llevó. Esa es la manera que seleccionó el equipo, para trackear el esfuerzo individual.

Cuando una *user story* queda completada, como se cuenta con la información del esfuerzo, el equipo puede conocer la relación entre el esfuerzo real, y el esfuerzo estimado.

La comparativa entre lo estimado, y lo realmente desarrollado, le va a otorgar al equipo lecciones aprendidas a medida que se avanza *sprint* a *sprint*.

Palabras clave



Total Effort

1

Ilustración 23 - Atributo "Total Effort" donde se lleva el registro del esfuerzo en Jira para cada la actividad

6.3.2. Descripción de los *sprints*

Se establecieron objetivos y propósitos para cada *sprint*, junto con la documentación de las fechas actividades, y el esfuerzo invertido. A continuación (Ilustración 24), se presenta un ejemplo ilustrativo:

Sprint	1
Fecha	Desde el 17/6/2024 hasta el 1/7/2024
Objetivos	Determinar la velocidad promedio inicial del equipo. Completar CRUD de entidades clave: Monedas, productos, tipos de productos, y proveedores. Desarrollar una versión preliminar del sistema para ser utilizada con clientes HTTP como Swagger o Postman. Evaluar la química del equipo en las primeras actividades de desarrollo.
Esfuerzo	45 SP (equivalentes a 45 horas hombres, lo cual fue acordado por el equipo)
Entregable	Si bien no se generó un release para el cliente, se considera que se realizan releases internos dentro del equipo. El equipo consideró que, al culminar este sprint, se cuenta con una versión preliminar básica del sistema, el cual sirve para ser utilizado por un cliente HTTP como, por ejemplo: Swagger o Postman. El entregable de este sprint contempla los objetivos del mismo, los cuales fueron mencionados anteriormente en esta tabla.

Ilustración 24 - Objetivos y propósitos de *sprint* (Ejemplo)

Evidencia de la Scrum *retrospective* de dicho *sprint* (Ilustración 25):

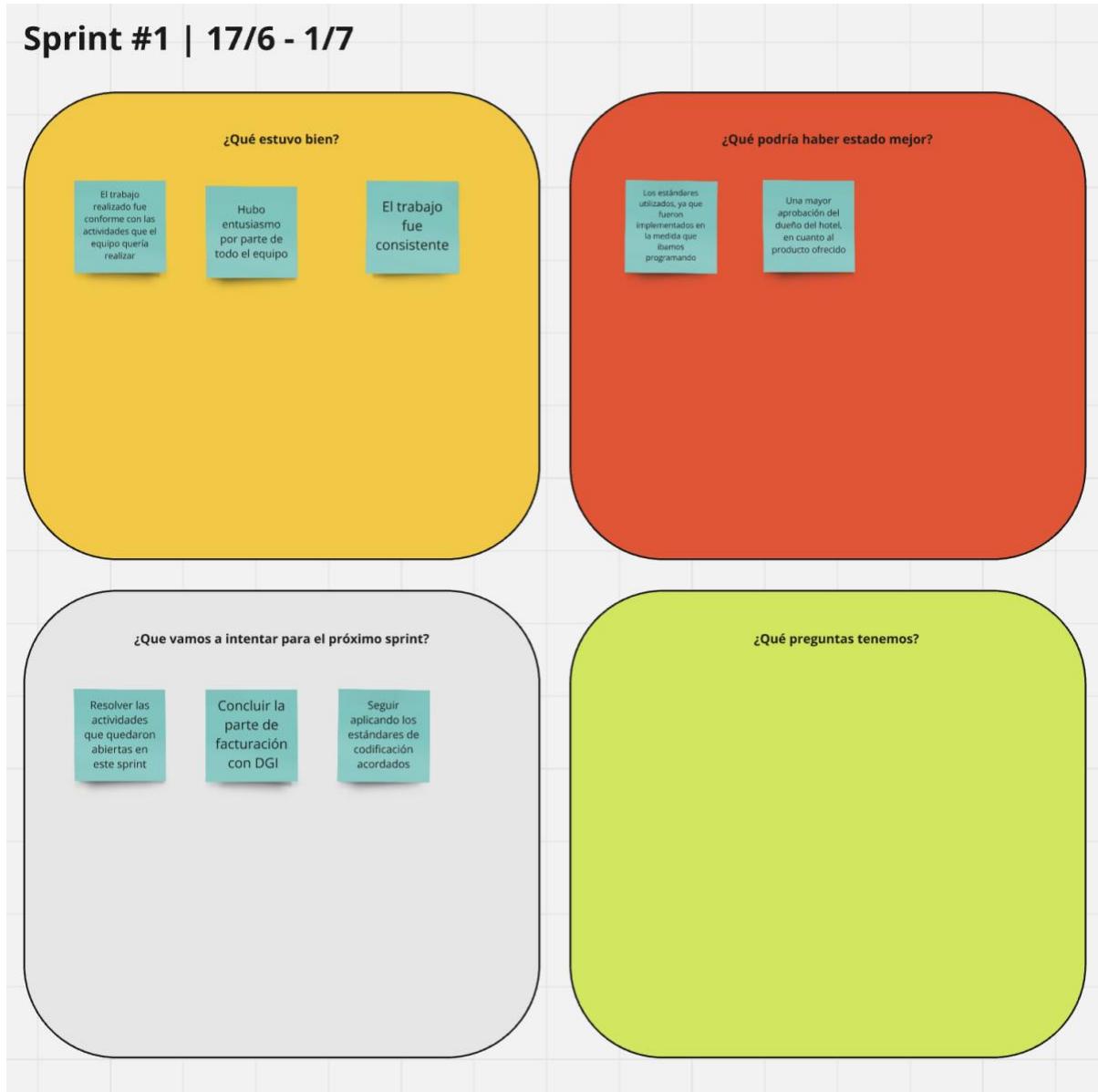


Ilustración 25 - *Sprint retrospective* del *sprint* #1

El detalle completo de cada *sprint* se puede visualizar en los anexos comprendidos entre el anexo [Documentación del Sprint #0](#) hasta el anexo [Documentación del Sprint #9](#).

6.3.3. Métricas de gestión

6.3.3.1. Esfuerzo total

El esfuerzo total del proyecto, sumando desde el *sprint 0* hasta el *sprint 9*, muestra que el equipo estimó un total de 696 *story points*. En cuanto al esfuerzo real, incluyendo el *sprint 0*, el equipo implementó un total de 629.5 *story points*.

El *sprint 0* se desglosa en las siguientes categorías: 15 *story points* fueron dedicados a la corrección de bugs, 61 *story points* al *refactor*, 346.5 *story points* al desarrollo, 201 *story points* a la documentación y 6 *story points* a la investigación. Esto resulta en un total de 48.5 *story points* de esfuerzo real para el *sprint 0*, que se suma al total de los *sprints* restantes.

Los cálculos realizados constan de la suma de las columnas "Estimación en SP" y "Esfuerzo real en SP", de las tablas que hay en las secciones "*Review*" del análisis de cada *sprint*. Dichas tablas, se pueden visualizar en los anexos [Documentación del Sprint #0](#) hasta el anexo [Documentación del Sprint #9](#).

6.3.3.2. Velocidad del equipo

La velocidad se mide en puntos de historia completados por *sprint*. Esta métrica ayuda al equipo a predecir cuánto trabajo puede asumir en futuros *sprints* basándose en el rendimiento anterior.

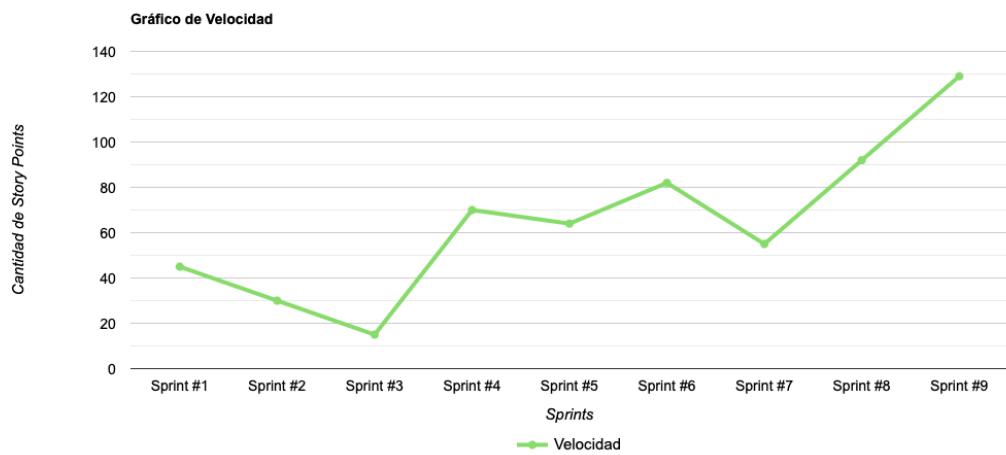


Ilustración 26 - Gráfica de velocidad

A lo largo de los nueve *sprints* de desarrollo (Ilustración 26), se puede observar una fluctuación en la velocidad del equipo en lo que corresponde a *story points*. En los primeros *sprints*, la tendencia muestra una velocidad moderada, con el *sprint 2* siendo impactado por la carga académica de uno de los estudiantes, quien estaba cursando la materia de Analítica para

Datawarehousing y desarrollando el obligatorio. Este compromiso afectó la dedicación de dicho integrante, lo que se reflejó en la capacidad productiva del equipo.

Durante el *sprint* 3, se puede observar la mayor disminución en la velocidad, siendo el *sprint* de menor esfuerzo del proyecto. Además del tiempo invertido en la preparación de la primera revisión con Marcelo Cagnani, dos de los integrantes del equipo tuvieron que realizar un examen de la materia Arquitecturas Empresariales, dictada por la tutora Helena Garbarino, lo cual claramente influyó en el rendimiento de esos integrantes durante ese periodo.

En lo que corresponde a los últimos dos *sprints*, se puede observar fácilmente que el equipo aumentó en gran medida la velocidad, lo cual es comprensible dado que estos son los *sprints* finales reflejan el esfuerzo por finalizar el proyecto y realizar la mayor cantidad de trabajo posible antes de la entrega. Este aumento en la velocidad también se vio beneficiado por el hecho de que dos de los estudiantes que forman parte del equipo, tomaron días de licencia por estudio, lo cual permitió que se le pueda dedicar mayor cantidad de horas al proyecto.

A pesar de los altibajos que se pueden apreciar, la tendencia general indica que el equipo se adaptó a las circunstancias, maximizando su esfuerzo en los momentos de mayor criticidad y gestionando los compromisos de los integrantes en paralelo con el avance del proyecto.

La velocidad final del equipo (en *story points*) se puede calcular así: (Velocidad *Sprint*#1 (45) + Velocidad *Sprint*#2 (30) + Velocidad *Sprint*#3 (15) + Velocidad *Sprint*#4 (70) + Velocidad *Sprint*#5 (64) + Velocidad *Sprint*#6 (82) + Velocidad *Sprint*#7 (55) + Velocidad *Sprint*#8 (91.5) + Velocidad *Sprint*#9 (128.5)) / Cantidad de *Sprints* (9) = 64.5 *story points* promedio por *sprint*.

Los cálculos realizados para determinar la velocidad, también se pueden encontrar en la sección "Velocidad" del análisis de cada *sprint*. Dichas tablas, se pueden visualizar en los anexos [Documentación del Sprint #1](#) hasta el anexo [Documentación del Sprint #9](#).

6.3.3.3. Desvío de estimaciones

La variación en las estimaciones se define como la diferencia entre los puntos de historia que el equipo preveía completar durante un *sprint* y aquellos que efectivamente se pudieron cumplir. Una diferencia entre estos valores puede significar la necesidad de revisar las estimaciones previas o la aparición de trabajo extra que no estaba planificado, lo cual influye en la productividad de los integrantes equipo. El hecho de evaluar esta variación/desvío, permite al equipo optimizar el enfoque de cómo se planifican y administran los *sprints*, mejorando así el rendimiento en proyectos futuros.

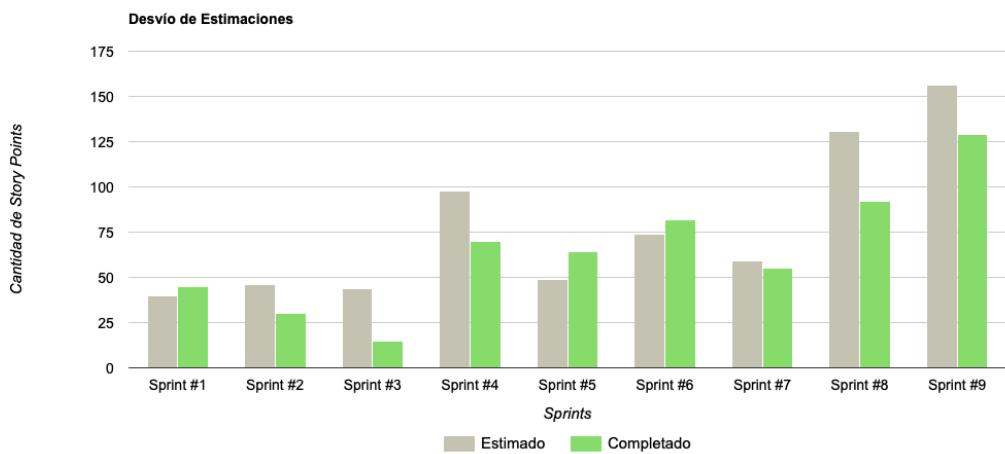


Ilustración 27 - Gráfica de desvío de estimaciones

Las columnas grises se corresponden con la cantidad de *story points* estimados de cada *sprint*. Las columnas verdes se corresponden con la cantidad de *story points* de esfuerzo real de cada *sprint*. (Ilustración 27)

A lo largo del proyecto, se identificaron varias razones que explican las desviaciones entre lo estimado y lo completado en cada uno de los *sprints*. En primer lugar, se adoptó un enfoque conservador en las estimaciones, optando por sobreestimar el esfuerzo necesario con el fin de evitar que quedara trabajo sin completar al finalizar un *sprint*. Esta fue la primera ocasión en la que el equipo asumió la responsabilidad total de las estimaciones, ya que, en proyectos anteriores (en los respectivos trabajos de cada uno de los integrantes del equipo), aunque se empleaba Scrum, la intervención en este proceso no era tan decisiva como en este caso.

En algunos *sprints*, como el *sprint 5*, se observó que se completó más trabajo de lo estimado, debido a que las tareas resultaron ser menos complejas de lo previsto inicialmente. Sin embargo, en otros *sprints*, como el *sprint 3*, se realizó menos de lo estimado debido a compromisos externos, como exámenes y preparación de revisiones, factores que no fueron considerados en las estimaciones iniciales y afectaron la capacidad de cumplir con los objetivos planteados.

Además, la falta de precisión en las estimaciones también se debió a que, en varias ocasiones, se enfrentaron tareas nuevas. Por ejemplo, al desarrollar el primer CRUD para la aplicación *web frontend*, la estimación fue imprecisa, pero en posteriores ocasiones, al poder reutilizar partes del código, las estimaciones se ajustaron mejor a la realidad, reduciendo así el tiempo y esfuerzo requerido.

Otra causa de las desviaciones fue la necesidad de profundizar en los requerimientos del proyecto para entender cómo aplicar las reglas de negocio del hotel. A menudo, este análisis requirió más

tiempo de lo previsto. En estas ocasiones, Francisco Cabanillas, ingeniero en requerimientos del equipo y conocedor del hotel, brindó asistencia aclarando los requisitos para que pudieran ser aplicados correctamente. Este proceso adicional influyó en los tiempos de entrega.

También hubo decisiones técnicas que afectaron las estimaciones, como ocurrió con la implementación de la funcionalidad de automatización de la obtención de facturas mediante FacturaLista. Durante este proceso, se debió investigar y tomar una decisión entre utilizar el *SOAP* o *REST* de la empresa, lo cual generó retrasos adicionales. Además, la complejidad de implementar las reglas de negocio del hotel a través de estos servicios prolongó aún más el desarrollo.

Por último, las estimaciones se vieron afectadas por el hecho de que dos integrantes del equipo compaginaban su trabajo con el proyecto. Esto a veces impidió que se alcanzaran los objetivos previstos, ya que no siempre fue posible equilibrar ambas responsabilidades. En los últimos dos *sprints*, la diferencia entre lo estimado y lo completado fue mayor debido a que se agregaron tareas adicionales por encontrarse en la fase final administrativa del proyecto, necesaria para la entrega en el portal de gestión. No obstante, las tareas pendientes de finalizar continuarán en los *sprints* 10 en adelante, fuera del alcance de la entrega administrativa.

Los cálculos realizados para poder determinar las desviaciones constan de la suma de las columnas "Estimación en SP" y "Esfuerzo real en SP", de las tablas que hay en las secciones "*Review*" del análisis de cada *sprint*. Dichas tablas, se pueden visualizar en los anexos [Documentación del Sprint #1](#) hasta el anexo [Documentación del Sprint #9](#).

6.3.3.4. Uso de burndown chart

El gráfico de *burndown* (Ilustración 28) es una herramienta esencial para el equipo, debido a que es crucial para monitorear los ítems restantes en el *backlog* al comenzar (y durante) cada *sprint*. Ofrece una visión clara del avance del equipo y es fundamental para asegurar que los objetivos del *sprint* se cumplirán en el tiempo previsto.

Cualquier desviación puede ser rápidamente identificada gracias a este gráfico. JIRA, además, facilita una representación visual que actualiza constantemente el estado del trabajo, mostrando el progreso, lo que se ha completado y lo que aún está pendiente, lo que es vital para la gestión eficaz del *sprint*.

Informe de trabajo completado

[Cómo leer este informe](#)



Ilustración 28 - Ejemplo de gráfico burndown

6.3.3.5. Retrabajo

Los errores detectados en el proyecto, JIRA permite que se etiqueten claramente como "bug". Esta clasificación permite a los miembros del equipo identificar rápidamente las tareas que requieren correcciones y asegura que se aborden de una manera prioritaria. Esta práctica ayuda a mantener un registro organizado de todos los problemas que necesitan resolverse durante el desarrollo del software. Además, el *refactor* u optimización también se considera retrabajo (no solo los *bugs*).

Cada vez que surge un problema, ya sea por fallos en las herramientas o por complicaciones con las integraciones del sistema, se genera una nueva tarea en JIRA correspondiente al bug. Esta tarea se maneja con un enfoque específico para corregir el error, permitiendo una solución estructurada y eficiente.

Este método no solo ayuda a organizar los problemas, sino que también proporciona al equipo un histórico detallado de las incidencias y sus soluciones, fundamental para la mejora continua del proyecto.

Para calcular el porcentaje de retrabajo, se sumó el esfuerzo total realizado por el equipo, en lo que corresponde a los *sprints* que van desde el primero hasta el noveno (no contamos el *sprint 0* por no ser de desarrollo). A ese resultado se le restó el esfuerzo total que el equipo destinó a corrección de *bugs* y *refactor*. Finalmente se pasa ese resultado a porcentaje.

A continuación, la lógica del cálculo:

- 629 SP (esfuerzo total de los *sprints* 0-9).
- 201 SP (esfuerzo de documentación)
- 6 SP (esfuerzo de investigación)
- 15 SP correspondientes al esfuerzo total de tareas de Jira vinculadas a la corrección de *bugs*. Estas tareas se pueden observar en las tablas que hay en las secciones "Review" del análisis de cada *sprint*. Dichas tablas, se pueden visualizar en los anexos [Documentación del Sprint #0](#) hasta el anexo [Documentación del Sprint #9](#).
- 61 SP correspondientes al esfuerzo total de tareas de Jira vinculadas a *refactor*. Estas tareas se pueden observar en las tablas que hay en las secciones "Review" del análisis de cada *sprint*. Dichas tablas, se pueden visualizar en los anexos [Documentación del Sprint #0](#) hasta el anexo [Documentación del Sprint #9](#).
- Si 629 se toma como el 100%, luego de restarle los 201 SP de documentación, y los 6 SP de investigación, queda en 422 SP.
- A esos 422, se tiene que calcular entonces el porcentaje que corresponde a SP de bugs y *refactor*, en este caso 76 (15 SP + 61 SP) es aproximadamente el 18.01%.
- El equipo concluye que el 18.01% aproximadamente del esfuerzo total, correspondió al arreglo de errores y *refactor*.

Se puede apreciar que el índice de retrabajo no fue bajo. Lo que castigó más al equipo en este sentido, fueron los dos grandes *refactor* que hubo que hacer. Uno correspondiente a cambios a nivel de toda la *API*, para cumplir con los estándares y buenas prácticas que el equipo había acordado. El segundo gran *refactor* vino de la mano de la funcionalidad de captura de facturas electrónicas desde FacturaLista.

Esta situación anteriormente descrita fue inevitable, y sucedió mientras el proyecto se encontraba medianamente avanzado en el caso del primer gran *refactor*, y muy avanzado en el caso del segundo gran *refactor*.

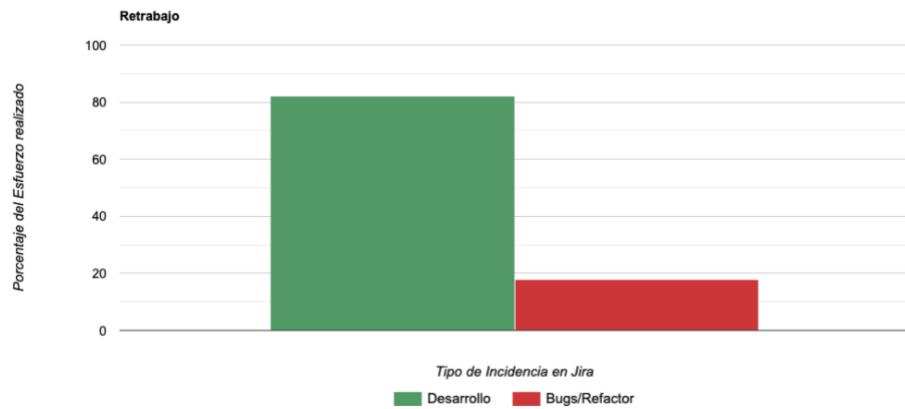


Ilustración 29 - Gráfica de retrabajo

Si observamos la gráfica (Ilustración 29), la columna verde representa el porcentaje de esfuerzo contenido en incidencias en Jira de tipo tarea/*task* o *user story*. La columna roja representa el porcentaje de esfuerzo contenido en incidencias en Jira de tipo bug.

6.3.4. Releases

Para este proyecto, se estableció que no habría *releases* intermedios hasta la salida en producción oficial, pactada con el cliente para el 1 de diciembre. La decisión de no realizar entregas intermedias fue acordada debido a que el hotel inicia su temporada a mediados de noviembre, la cual se extiende hasta finales de abril.

Esto permitió al equipo enfocarse en desarrollar el producto de manera continua y sin interrupciones, cumpliendo con todos los requerimientos y funcionalidades acordadas para la fecha de producción. Desde el inicio, se mantuvo una planificación clara para garantizar que el sistema estuviera completamente listo para su lanzamiento, coincidiendo con el inicio de la temporada del hotel.

A pesar de que no hubo entregas intermedias, se establecieron objetivos medibles dentro de cada *sprint* para asegurar el progreso adecuado del proyecto y cumplir con las expectativas del cliente para la fecha final.

6.4. Gestión de la comunicación

Desde el inicio del proyecto, se decidió establecer un enfoque claro y eficiente para la comunicación, tanto entre los miembros del equipo y con la tutora, como con los interesados externos. A continuación, se explica la gestión de comunicación con los interesados. En el anexo [Gestión de la comunicación](#) se detalla la misma con el equipo y con la tutora.

6.4.1. Comunicación con los interesados

En esta etapa se asignaron responsables para la comunicación con cada interesado. Francisco Cabanillas, como *project manager*, se comunica con Maximiliano Correa del Hotel Tío Tom, quien es el *product owner* del proyecto. Andrés Quintero se encarga de la comunicación con Diego Dodel (representante de FacturaLista). Las herramientas principales utilizadas para la comunicación dentro del equipo son WhatsApp, Google Meet y Microsoft Teams.

Las reuniones semanales de Scrum (*daily*) y las reuniones con la tutora Helena Garbarino se realizan a través de Microsoft Teams. Estas reuniones son fundamentales para garantizar la alineación del equipo y la correcta ejecución del proyecto.

Con relación a la comunicación con el Hotel Tío Tom, Francisco Cabanillas se encarga de coordinar las reuniones y mantener una comunicación constante con Maximiliano Correa a través de WhatsApp. Esto permite asegurar que el *product owner* esté siempre informado sobre el progreso y pueda proporcionar retroalimentación o ajustes necesarios en tiempo real.

Por otro lado, Andrés Quintero gestiona la comunicación con Diego de FacturaLista, manteniendo el contacto principalmente por Google Meet y correo electrónico. Esta comunicación es clave para coordinar la integración de los sistemas y asegurar que las funcionalidades desarrolladas cumplen con los requerimientos del cliente.

Las reuniones con la tutora Helena Garbarino se llevan a cabo de manera regular para revisar el avance del proyecto y obtener su guía en aspectos técnicos y de gestión. Estas reuniones suelen finalizar con la programación de la siguiente sesión, asegurando una continuidad en la supervisión y apoyo al equipo.

La estrategia de comunicación establecida ha permitido una interacción efectiva y fluida entre todos los interesados, lo cual ha sido crucial para mantener el proyecto dentro de los plazos establecidos y con la calidad esperada.

6.5. Gestión de riesgo

La gestión de riesgos es necesaria en cualquier proyecto, incluyendo uno de tres integrantes, porque ayuda a anticipar y mitigar posibles problemas que podrían surgir durante el desarrollo del sistema.

Esto permite planificar mejor y asignar recursos de manera eficiente para asegurar que los aspectos más importantes del proyecto reciban la atención necesaria. Además, al identificar y mitigar riesgos, se reducen los impactos negativos que estos puedan tener, lo que mejora la toma de decisiones y la eficiencia del equipo.

Todo esto contribuye a aumentar las probabilidades de entregar un sistema de calidad dentro del plazo y presupuesto acordados, lo que, en última instancia, garantiza la satisfacción del cliente. En resumen, la gestión de riesgos es clave para asegurar el éxito y la calidad en la entrega del proyecto.

Este proceso constó de cuatro etapas: identificación de riesgos, análisis de riesgo, clasificación de riesgo y monitorización de riesgos. A medida que estos eran identificados y analizados, se clasificaban en uno de los tipos de riesgo que se optaron: Proyecto, Producto, Equipo, Técnico y Externo.

En la etapa de análisis, para cada riesgo se fijaba su descripción, probabilidad de ocurrencia, impacto y un plan de mitigación. La probabilidad de ocurrencia se evaluaba del 1 al 10 (siendo 1 una probabilidad baja y 10 la probabilidad más alta) y el impacto se medía en función de cómo

afectaría al proyecto, clasificándose también del 1 al 5 (siendo 1 un impacto leve y 5 un impacto grave). A partir de la multiplicación de estos valores fijados se obtenía el nivel de riesgo asociado que tendría. Estos riesgos luego se visualizaban en matriz de probabilidad x impacto, conocida por el equipo como matriz de riesgo (Ilustración 30).

Los planes de mitigación incluyen acciones específicas para reducir la probabilidad de que ocurran los riesgos o minimizar su impacto en caso de que se materialicen.

Una vez completadas estas etapas, se estableció un sistema de monitorización de riesgos cada dos *sprint* (cuatro semanas corridas) que permitía realizar un seguimiento continuo del estado de cada riesgo identificado y la efectividad de las medidas de mitigación implementadas. Esta monitorización aseguraba que el equipo estuviera preparado para responder rápidamente ante cualquier cambio en el perfil de riesgo del proyecto, facilitando la adaptación y la toma de decisiones informadas a lo largo del desarrollo.

6.5.1. Descripción de los tipos de riesgo

Riesgo de proyecto: Riesgos relacionados con la planificación del proyecto, la estimación de tareas, la gestión del alcance y la distribución de la carga de trabajo.

Riesgo de producto: Riesgos relacionados con la calidad del producto, incluyendo defectos de software, usabilidad y ergonomía, así como también cambios en los requerimientos del cliente.

Riesgo de equipo: Riesgos relacionados con la comunicación entre los miembros del equipo, tutores, clientes y otros interesados.

Riesgo técnico: Riesgos relacionados con dificultades técnicas, desconocimiento de tecnologías y problemas de implementación.

Riesgo externo: Riesgos que provienen de factores externos al proyecto, como regulaciones gubernamentales, proveedores, problemas personales de fuerza mayor y otros terceros.

6.5.2. Matriz de riesgo (probabilidad e impacto)

Matriz de riesgo	1	2	3	4	5	Impacto
Probabilidad						
1	1	2	3	4	5	5
2	2	4	6	8	10	10
3	3	6	9	12	15	15
4	4	8	12	16	20	20
5	5	10	15	20	25	25
6	6	12	18	24	30	30
7	7	14	21	28	35	35
8	8	16	24	32	40	40
9	9	18	27	36	45	45
10	10	20	30	40	50	50

Ilustración 30 - Matriz de riesgo

En esta se definen los rangos de criticidad como:

- Nivel crítico bajo: 1 – 14. No requiere atención.
- Nivel crítico medio: 5 – 30. Requiere atención. Analizar el uso del plan de mitigación.
- Nivel crítico alto: 31 – 50. Requiere atención urgente. Utilizar plan de mitigación.

6.5.3. Riesgo tipo

Riesgo	TEC-02
Título	Dificultades con la implementación con DGI u intermediarios
Tipo	Técnico
Descripción	Existe el riesgo de encontrarnos con dificultades para implementar algún detalle con las tecnologías por no ser especialistas de estas
Plan de mitigación	Investigar documentación disponible, realizar pruebas y de ser necesario contactar al área de ayuda técnica de la DGI.
Probabilidad (al inicio)	6
Impacto (al inicio)	4
Nivel de riesgo promedio	34

Ilustración 31 - Ejemplo de riesgo tipo

Riesgo	PRJ-01
Título	Tiempos de dedicación personal al proyecto
Tipo	Proyecto
Descripción	No lograr cumplir con los tiempos mínimos necesarios de dedicación personal al proyecto.
Plan de mitigación	Comunicar con anticipación las dificultades para coordinar cobertura de los requerimientos por parte de otros miembros
Probabilidad (al inicio)	8
Impacto (al inicio)	4
Nivel de riesgo promedio	37

Ilustración 32 - Ejemplo 2 de riesgo tipo

6.5.4. Evolución en el tiempo

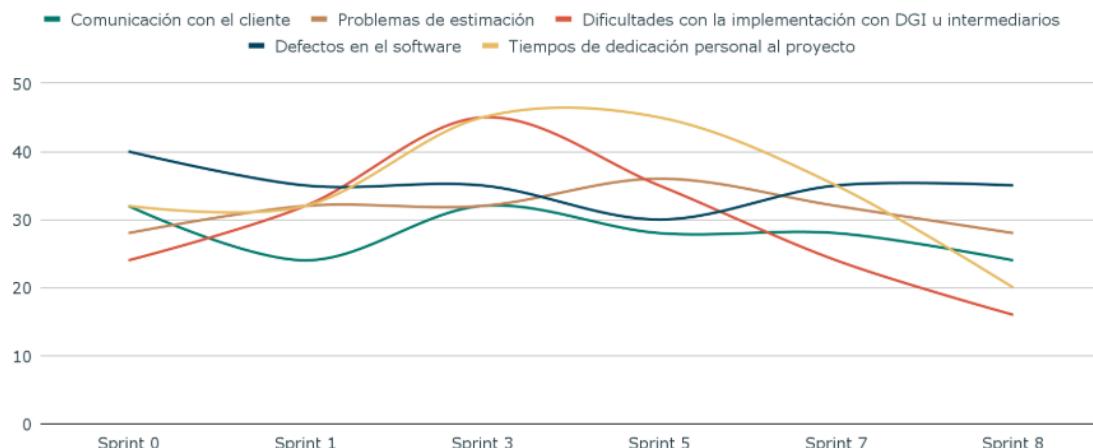


Ilustración 33 - Grafica evolución de los principales riesgos

Estos cinco riesgos que se detallan (Ilustración 33) fueron los principales del proyecto por el rol que cumplían en el resultado final del producto y lo alto que fue su nivel de riesgo en el tiempo.

EQP-02 Comunicación con el cliente: Este riesgo se relaciona con una comunicación interrumpida o incompleta con algún referente del hotel, esto ocasionaría dificultades para cumplir con el plan de calidad. Por experiencias pasadas de Francisco con el hotel el equipo esperaba grandes dificultades para comunicarse con el cliente, especialmente en el periodo de abril a septiembre que el hotel no se encuentra abierto, este únicamente funciona en verano como se comentó previamente.

De todos modos, Maximiliano se encontró disponible la mayoría de las veces que fue necesario y considerando el conocimiento del hotel y el prototipo durante el periodo de desarrollo no fue necesaria una comunicación periódica.

PRJ-01 Problemas de estimación: Se esperaban dificultades al inicio del proyecto debido a la falta de experiencia del equipo en la estimación de tiempos y recursos. Esto resultaba en

proyecciones optimistas que no reflejaban la realidad de las tareas requeridas, lo que llevó a la necesidad de incluir un alcance menos optimista por sprint.

TEC-02 Dificultades con la implementación con DGI u intermediarios: Este riesgo refiere a cualquier retraso, desinformación y demoras en el proceso de integración del ingreso automático de facturas. La información brindada en la página de la DGI no es reciente ni muy clara. Muestra grandes tablas con atributos varios, así como también *responses* en formato XML muy extensos sin contexto ni descripción.

Para minimizar este riesgo se optó tomar el camino de realizar la integración por un intermediario, en este caso Factura Lista quien era proveedor del hotel desde antes. Se apuró una reunión con la empresa para poder iniciarla sin retrasos.

PRD-03 Defectos en el software: Los defectos se van a presentar en cualquier software, lo que importa es cuanto se puedan minimizar principalmente los riesgos que puedan implicar estos. Para esto se planificó un trabajo minucioso de *test* unitarios en el código y manejo de excepciones que puedan capturar los defectos sin dejar fuera de servicio el sistema.

PRJ-02 Tiempos de dedicación al proyecto: El tiempo es un factor principal en este proyecto que tiene un periodo acotado y todos los integrantes trabajamos a tiempo completo. El seguimiento de los tiempos de dedicación al proyecto reveló que el equipo iba a tener que emplear más horas de las inicialmente previstas. Esto debido a varios factores, como las dificultades en la estimación de tiempos, la aparición de defectos en el software y la necesidad de realizar ajustes en la integración con la DGI.

Para mitigarlo se mantuvo una comunicación constante en el equipo ante cualquier contratiempo para buscar soluciones y algunos miembros del equipo redujeron las horas de sus trabajos a tiempo completo para poder dedicarle más tiempo y enfoque a este proyecto.

7. Gestión de calidad

7.1. Aseguramiento de la calidad

En el transcurso de ese capítulo se discutirán las medidas que el equipo tomó para asegurar la calidad tanto del proceso como del producto generado. También se describirán conceptos, estándares y métricas relacionadas a la calidad.

7.2. Objetivos de calidad

Previo a la definición de los objetivos de calidad que fueron propuestos para este proyecto, se considera importante compartir lo que el equipo entiende por calidad, ya que los objetivos definidos están estrechamente ligados con este concepto.

La visión de calidad del equipo comienza con una comprensión profunda de las necesidades del cliente, estableciendo objetivos claros para cumplirlas e incluso superar sus expectativas. No se trata únicamente de entregar un producto terminado, sino de asegurar que cada aspecto del desarrollo esté alineado con los requerimientos establecidos.

Para el equipo, la calidad abarca tanto el producto como el proceso que lo crea. Va más allá de simplemente cumplir con los requisitos; implica un enfoque desde todos los puntos en cada etapa del desarrollo, garantizando que el resultado final sea excepcional en todos los sentidos. Por eso, hemos dividido nuestros objetivos de calidad en dos grandes categorías.

7.2.1. Objetivos de calidad del producto

Desde esta perspectiva definida, el equipo se enfocará en crear un producto de alta calidad que cumpla con todas las funcionalidades esenciales y prioritarias establecidas en el alcance del proyecto. Se aspira a lograr un alto nivel de satisfacción tanto del cliente como de los usuarios a través de revisiones y validaciones exhaustivas. Además, se asegurará que el producto final sea funcional y adecuado como base para futuros emprendimientos.

Dentro de esta definición, se contemplan los siguientes puntos:

- Lograr que el cliente se sienta a gusto y cómodo utilizando el sistema, generando su confianza.
- Eliminar los errores derivados del factor humano, especialmente en la carga de facturas.
- Entregar el producto en un ambiente de producción.
- Entregar un producto sin defectos críticos y bloqueantes para el cliente.

7.2.2. Objetivos de calidad del proceso

En lo que respecta al proceso, el equipo se compromete a entregar trabajos de calidad y a cumplir con lo planificado a lo largo de todo el proceso. Esto incluirá tanto el desarrollo de la solución como la documentación, asegurándose de seguir las actividades de calidad en cada fase de acuerdo con el plan y los estándares establecidos.

Dentro de esta definición, se contemplan los siguientes puntos:

- Definir de forma clara y sin lugar para ambigüedades los requerimientos del sistema.
- Al finalizar el producto y realizar la entrega correspondiente, obtener la validación del cliente.
- Trabajar dentro del marco Scrum, con las adaptaciones y personalizaciones necesarias para la organización del trabajo y los procesos del equipo.
- Lograr que las horas dedicadas a la resolución de *bugs* y refactorizaciones por deficiencias detectadas no superen el 10% de las horas totales dedicadas al desarrollo del sistema.
- Detectar y documentar los riesgos relacionados al proyecto para poder actuar en consecuencia, así como reaccionar ante los no contemplados inicialmente.

7.2.3. Justificación de objetivos de calidad

Los objetivos de calidad previamente definidos no fueron seleccionados al azar, sino que se argumentan las siguientes razones para su elección:

- **Importancia:** Se definieron objetivos que impactan directamente en la experiencia del cliente, así como en el producto final que se entregará. Todos ellos fueron establecidos en conformidad con el cliente. Además, como equipo, también se definieron objetivos internos para garantizar una entrega de calidad, con un enfoque tanto en el producto como en el proceso.
- **Contexto:** Dado el tiempo de duración del proyecto y su complejidad, este se define como un *MVP* (mínimo producto viable) para satisfacer las necesidades del cliente, como ya se mencionó anteriormente. Por esta razón, el equipo identificó estos objetivos como los más alcanzables dentro del plazo del proyecto.

Se considera que esta selección de objetivos permitirá lograr tanto un proceso como un producto de calidad, impactando directamente en el entregable que se generará para el cliente.

Es importante destacar que, aunque se reconocen otros aspectos en los que se podrían haber establecido objetivos, se decidió reservarlos para la siguiente etapa del proyecto.

7.2.4. Atributos de calidad

Adicionalmente a los objetivos de calidad de producto y proceso descritos anteriormente, una vez definidos los requerimientos no funcionales del sistema, se establecieron los atributos de calidad necesarios para satisfacerlos.

La estructura de esta definición se compone de la siguiente manera:

RNF-X – Nombre de RNF

1. **Atributo de calidad:** Nombre de atributo de calidad.
 - a. **Definición:** Definición del atributo de calidad.
 - b. **Recomendaciones:** Recomendaciones para cumplir con el atributo de calidad
 - i. Recomendación 1.
 - ii. Recomendación 2.
 - iii. Recomendación 3.

Para ver el listado completo junto a qué las recomendaciones indicadas según cada atributo de calidad, referirse a los anexos ubicados entre [Documentación del Sprint #0](#) y el [Documentación del Sprint #9](#).

7.3. Planificación de la calidad

Dentro del equipo, se asignó a Andrés Quintero la responsabilidad de gestionar y organizar las actividades relacionadas con la calidad.

Como primera medida, se elaboró un plan de calidad alineado con los objetivos establecidos, que sirve como guía a lo largo del proyecto. El propósito de este plan es asegurar la calidad en todos los aspectos necesarios del proyecto y en los productos resultantes.

El plan detalla todas las actividades a realizar, las técnicas sugeridas para llevarlas a cabo, los roles de los participantes en cada una y los productos que cada actividad consume y genera como resultado. Este plan se organiza en diferentes tipos de actividades que se llevan a cabo durante el proyecto: empatía, ingeniería de requerimientos, diseño, desarrollo y pruebas. Además, se definen actividades de apoyo, independientes de las mencionadas anteriormente.

A continuación, se ofrece un breve resumen de cada fase:

Empatía

En esta fase, el enfoque se centró en comprender las necesidades y restricciones tanto del cliente como del usuario final. Se realizarán reuniones y entrevistas para recolectar esta información y se estudiarán sistemas similares existentes para obtener un análisis comparativo.

Ingeniería de requerimientos

En esta fase, se especificarán y validarán los requerimientos del proyecto con el cliente y los usuarios. Se utilizarán técnicas de colaboración y revisión para asegurar que los requerimientos sean claros y completos, y se priorizará según la importancia para el proyecto.

Diseño y arquitectura

Una vez definidos y validados los requerimientos, será diseñada la solución técnica, incluyendo la arquitectura y los prototipos. También se evaluará y seleccionará las tecnologías adecuadas para el *backend* y el *frontend*, y serán definidos los estándares de codificación.

Desarrollo

En esta fase, el equipo se capacitará según sea necesario y comenzará a codificar los requerimientos en historias de usuario. Se seguirán los estándares de codificación definidos y se realizarán revisiones de código entre pares para asegurar la calidad.

Pruebas

La fase de pruebas incluirá el diseño y la ejecución de pruebas funcionales, unitarias, de integración, de carga, de usabilidad y de portabilidad para asegurar que todos los componentes del sistema funcionen correctamente y cumplan con los requerimientos.

Actividades de apoyo

Además de las fases mencionadas, se realizarán actividades de apoyo que son esenciales para la gestión y el aseguramiento de la calidad del proyecto, como la planificación de *sprints*, análisis de riesgos, y reuniones de revisión y retroalimentación.

El plan de calidad puede consultarse en el siguiente anexo: [Plan de Calidad](#)

7.4. Proceso de calidad

En esta sección, serán definidos los estándares con los que el equipo deberá cumplir, así como también las herramientas de apoyo.

7.4.1. Estándares de calidad

Estándares de documentación

Documento	Estándar
302	Formato general de los trabajos finales de carrera.
303	Especificación de aspectos gráficos y de estructura a ser tomados en cuenta durante la corrección de los trabajos finales de carrera.
304	Procedimiento que seguir durante el desarrollo de los trabajos finales de carrera.
UML	Diagramación UML

Estándares de codificación

Para la codificación de todos los módulos del sistema, se siguieron los lineamientos oficiales establecidos de cada una de las tecnologías utilizadas.

Tecnología	Estándar
API	<i>REST</i> [16]
	Guías de diseño de <i>RESTful APIs</i> de Microsoft [16]
TypeScript	Guía de estilo de Google [17]

JavaScript	Guía de estilo de Google [18]
Flutter	Guía de estilo Flutter team [19]
.NET	Guía de estilo Flutter team [20]

Estándares de usabilidad

Como ya fue mencionado antes en este documento, el producto obtenido se acopla a las heurísticas de Nielsen. La finalidad de utilizar estos lineamientos como un estándar a seguir es reducir la curva de aprendizaje del sistema, así como también, que este sea intuitivo. Todo esto con foco en que el usuario final del sistema se sienta a gusto, confiado y cómodo para trabajar con el mismo de la manera más inmediata posible.

Definition of done

Como parte de la utilización del marco Scrum, fue necesario establecer una definición que formalice cuando una funcionalidad está pronta para su utilización, para poder cerrar el ciclo de desarrollo sobre la misma.

Esta definición, contiene los siguientes puntos a ser cumplidos:

- Código finalizado.
- Pruebas unitarias realizadas y con resultado exitoso.
- No hay defectos reportados en la funcionalidad.
- La funcionalidad se encuentra pronta para ser integrada a la rama estable.
- Se cuenta con la aprobación para dicha integración.

Las horas correspondientes al desarrollo de dicha funcionalidad fueron registradas en la aplicación de gestión (JIRA).

Estándares para el control de versiones

- Los pull requests se someten a revisiones cruzadas. Antes de ser aceptados, otro integrante del equipo los revisa para sugerir mejoras y prevenir errores.
- Utilizar las ramas main, develop, y cada integrante debe crear sus propias ramas a medida que desarrolla una nueva funcionalidad.
- La rama será eliminada una vez completado el merge a la rama principal.

- Cada nueva rama llevará un prefijo de feature, fix, o hotfix, acompañado del nombre de la incidencia reportada en JIRA a la cual hace referencia.

Estándares adicionales

Todo el código estará escrito en inglés, dado que es el idioma predominante en el ámbito del *software* y facilita la mantenibilidad si se incorporan nuevos desarrolladores.

Se seguirán a los principios de Clean Code, una referencia fundamental que estudiamos durante la carrera.

7.4.2. Revisiones

Revisiones internas

Al final de cada *sprint*, el equipo realizó reuniones retrospectivas (establecidas previamente por el [marco](#)) con la finalidad de analizar la performance del equipo.

Revisiones externas

En el transcurso de los meses que duró el proyecto, se realizó una instancia formal de revisión por parte de un profesional calificado perteneciente a la universidad. En esta oportunidad, por medio de nuestra tutora, fue sugerido Marcelo Cagnani.

El objetivo de esta revisión era presentar el estado de situación del proyecto a alguien externo y ajeno al proyecto, para así poder tener otra perspectiva. Esta instancia fue muy valiosa para el equipo, ya que le ayudó a identificar fortalezas y oportunidades de mejora, permitiéndole reaccionar a tiempo ante ciertas eventualidades.

Consulta con experto en arquitectura

Luego de la primera revisión externa, le fue recomendado al equipo realizar una consulta de asesoría junto a un experto en arquitectura. El profesor Gastón Mousqués fue el encargado de realizarla, quien es un referente dentro de la universidad.

En términos generales, el profesor se mostró de acuerdo con lo que el equipo expuso sobre la arquitectura del sistema. Las recomendaciones que realizó fueron para favorecer el desacoplamiento de los diferentes componentes del sistema.

El equipo tenía pensado utilizar una única base de datos, tanto para el negocio, como para el futuro módulo de *machine learning*. Gastón recomendó que haya al menos dos bases de datos. Una de ellas dedicada 100% al negocio, y la otra para favorecer el módulo de análisis de datos a futuro con *machine learning*. Debido a que las responsabilidades de ambas bases de datos son diferentes,

recomendó la separación. Esta recomendación favorece el cumplimiento del requerimiento no funcional de extensibilidad, en el anexo [RNF-7 - Extensibilidad](#) puede encontrar más información al respecto.

Otra recomendación realizada por Gastón, y que el equipo implementó (tal como sucedió con la anterior recomendación), es la de utilizar el patrón adapter para gestionar la integración con el servicio *SOAP* de Factura. El detalle de la implementación del patrón se puede observar en la descripción del módulo ExternalServices, en la sección [5.4.2.1.2](#).

7.4.3. Actividades de aseguramiento de la calidad

Como equipo, era muy importante definir un plan de pruebas robusto antes de comenzar el desarrollo del proyecto. El objetivo es tener procedimientos que permitan detectar defectos de manera temprana y priorizar su resolución de manera efectiva. Esto asegura que no se arrastren defectos severos hasta la producción.

El plan de pruebas también permite que los desarrolladores se concentren en las historias de usuario que están codificando, incluso si descubren un defecto durante el proceso, para registrar y resolver adecuadamente en el momento oportuno.

Como comentario, se tuvo en cuenta la utilización de la técnica TDD (*Test Driven Development*). Sin embargo, esta aproximación fue finalmente descartada luego de ser discutida dentro del equipo y con la tutora. El tiempo acotado de desarrollo, sumado a la poca experiencia del equipo con esta técnica, fueron factores clave para tomar esta decisión.

Son definidos varios tipos de pruebas que difieren en los aspectos de calidad que buscan asegurar y en la periodicidad de su ejecución. A continuación, se detallan los tipos de pruebas para tener en cuenta:

Pruebas automáticas

Las pruebas automáticas de código son una práctica fundamental en el desarrollo de *software* moderno, diseñadas para verificar automáticamente el comportamiento esperado de las aplicaciones y detectar posibles errores de manera temprana en el ciclo de desarrollo. Estas pruebas permiten evaluar si el código produce los resultados esperados bajo diversas condiciones y escenarios, contribuyendo así a mejorar la calidad del *software* y reducir los errores en producción.

En este proyecto, inicialmente el equipo se enfocará en implementar pruebas automáticas para el *backend*. Esto incluye tanto pruebas unitarias, que evalúan componentes individuales de código de manera aislada, como pruebas de integración, que verifican la interacción entre diferentes componentes del sistema. El objetivo es alcanzar un porcentaje de cobertura de pruebas superior al 90%, ya sea mediante pruebas directas o indirectas.

Las pruebas directas se centran en validar funcionalidades específicas a nivel de código, mientras que las pruebas indirectas aseguran que los diferentes módulos del sistema trabajen correctamente en conjunto.

Sugerencias:

Pruebas unitarias con moc para los *endpoints* de la *API* y la lógica de negocio.

Pruebas unitarias con base de datos "in memory" para la capa de acceso a los datos.

En cuanto al *frontend*, su incorporación dependerá de las condiciones y recursos disponibles. En caso de ser factible y adecuado para el proyecto, también buscaremos alcanzar altos estándares de cobertura de pruebas en esta capa del sistema.

Es válido aclarar que, el número objetivo a superar con las pruebas de cobertura fue basado en la cota a superar en los obligatorios de las materias Diseño de Aplicaciones 1 y Diseño de aplicaciones 2, materias que el equipo considera como fundamentales en la carrera y un buen *checkpoint* para la exigencia de este ítem en el proyecto.

Pruebas funcionales

Las pruebas de caja negra consisten en ejecutar manualmente un conjunto predefinido de casos de prueba funcionales. Su objetivo es detectar defectos en la funcionalidad del sistema, ya sea por errores de implementación o inconsistencias entre lo implementado y los requisitos definidos.

Las pruebas funcionales se ejecutarán antes de liberar cada uno de los *releases*. Se llevarán a cabo en un ambiente designado (*staging*, *test*, prueba), que será virtualmente idéntico al ambiente de producción. Esto asegurará que, si una funcionalidad funciona correctamente durante las pruebas funcionales, también funcionará en el ambiente de producción.

Cada caso de prueba funcional estará especificado con un identificador único, el identificador del requerimiento funcional asociado (si corresponde), una breve descripción del caso, el contexto necesario para su ejecución, la lista de pasos a seguir y el resultado esperado del mismo.

Pruebas de portabilidad

Las pruebas de portabilidad son un tipo de prueba diseñado para evaluar la capacidad de un *software* para ejecutarse de manera adecuada en diferentes entornos o plataformas.

El objetivo principal de estas pruebas es asegurar que la aplicación pueda ser instalada y ejecutada correctamente en diversos sistemas operativos, configuraciones de *hardware* y dispositivos. Esto incluye verificar que la aplicación mantenga su funcionalidad y desempeño esperados sin importar las variaciones en el entorno de ejecución.

Las pruebas de portabilidad se realizarán para asegurar que el *software* pueda ser instalado y ejecutado correctamente en una variedad de entornos y plataformas. Se verificará la capacidad del sistema para adaptarse a diferentes configuraciones de sistema operativo, *hardware* y redes, asegurando que mantenga su funcionalidad y desempeño esperados en cada escenario.

Pruebas de rendimiento

Las pruebas de rendimiento son pruebas diseñadas para evaluar cómo se comporta una aplicación bajo diversas cargas y condiciones de uso.

El objetivo principal de estas pruebas es medir y validar la capacidad del sistema para manejar la cantidad esperada de usuarios y transacciones, sin comprometer el rendimiento o la respuesta del sistema.

Las pruebas de rendimiento se llevarán a cabo para evaluar cómo se comporta el sistema bajo diversas cargas y condiciones de uso. Se simularán picos de uso y grandes volúmenes de datos para medir y validar la capacidad del sistema para manejar la cantidad esperada de usuarios y transacciones.

Se analizarán métricas clave como el tiempo de respuesta del sistema, la utilización de recursos y la escalabilidad, asegurando que el sistema pueda mantener un rendimiento óptimo durante su operación bajo carga

Pruebas de usabilidad

Las pruebas de usabilidad serán realizadas para evaluar la facilidad de uso y la experiencia del usuario con el *software*. Se diseñarán escenarios de prueba que simularán situaciones reales de uso, asegurando que los usuarios puedan completar tareas comunes de manera intuitiva y eficiente. Se medirá la navegación, la claridad de las instrucciones y la accesibilidad de las funciones clave del *software*.

Durante estas pruebas, se recogerán comentarios y observaciones de los usuarios para identificar áreas de mejora en la interfaz de usuario y en la experiencia general del usuario. El objetivo será optimizar la usabilidad del *software* para garantizar una interacción satisfactoria y productiva con los usuarios finales.

7.4.4. Gestión de incidentes

Para la gestión de los incidentes, fue elaborado un plan de gestión de incidentes para manejar de manera eficiente la aparición de defectos durante el desarrollo del proyecto. Este plan unifica el proceso para tratar cualquier tipo de defecto, independientemente de cómo sea reportado.

Reporte de incidentes:

Los incidentes podrán ser reportados por integrantes del equipo durante pruebas exploratorias, por integrantes del cliente durante pruebas en el ambiente de (*staging test*, prueba), o por usuarios finales del sistema

Para reportar un incidente, se creará un ticket en JIRA que contendrá información detallada para que cualquier integrante del equipo pueda entenderlo y determinar su prioridad. En caso de reportes por parte de terceros ajenos al equipo de desarrollo, estos podrán ser a través de otros medios para luego ser correctamente estructurados por un integrante del equipo.

Priorización y severidad:

Cada incidente será asignado a uno de los cinco niveles de severidad definidos por el equipo:

URGENTE: Implica un problema crítico, bloqueante, que debe ser resuelto de inmediato.

ALTA: Importante de resolver, pero no bloqueante para el equipo.

MEDIA: No urgente y no bloquea funcionalidades críticas.

BAJA: Incidente menor que no afecta significativamente al sistema.

La severidad determinará la prioridad de resolución del incidente.

Tratamiento de Defectos:

Los defectos serán tratados como tareas adicionales, similares al desarrollo de nuevas historias de usuario.

Al inicio de cada *sprint*, el equipo priorizará la resolución de defectos en base a las discusiones con el cliente y la severidad del defecto.

Los defectos de severidad alta podrán integrarse directamente al *sprint* actual si son considerados críticos para la funcionalidad existente, mientras que los urgentes serán resueltos lo antes posible independientemente del ciclo.

Seguimiento y resolución:

Se realizará un seguimiento continuo de los incidentes reportados para asegurar que sean resueltos en tiempo y forma.

Cada ticket en JIRA será actualizado con el progreso de la resolución y cualquier información relevante.

7.4.5. Métricas

Para garantizar y medir la calidad del producto generado, así como también de los procesos, fueron escogidas las siguientes métricas:

7.4.5.1. Métricas de Producto

Bugs reportados por sprint

- Medida: Número de *bugs* detectados durante un *sprint*.
- Forma de medir: Contabilizar el total de *bugs* reportados por *sprint*.

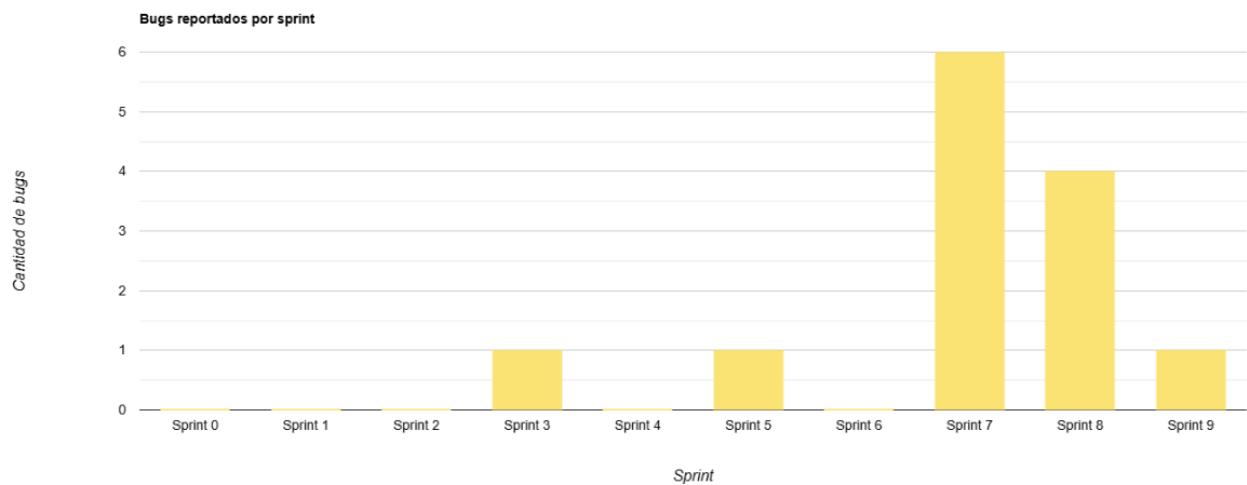


Ilustración 34 - Bugs reportados por sprint

En base a la gráfica presentada (Ilustración 34), la métrica de "*Bugs reportados por sprint*" nos da una visión sobre el número de *bugs* que se han detectado en cada ciclo de trabajo o *sprint*.

Podemos observar:

- *Sprints* iniciales (0 a 2): No se reportaron *bugs*, el equipo atribuye esto a que el proyecto estaba en fases tempranas.
- *Sprints* intermedios (3 a 6): Se reportaron pocos *bugs* (2 en total), lo que indica la introducción de nuevas funcionalidades y una detección de errores limitada, probablemente porque el desarrollo aún no había alcanzado una complejidad alta.
- *Sprints* finales (7 a 9): En estos *sprints* se observa un notable incremento en la cantidad de *bugs* reportados. Esto se atribuye a la incorporación de funcionalidades más complejas y al hecho de que, al estar el sistema más desarrollado, era más fácil detectar errores al probar el flujo completo de la aplicación.

Como acotación para finalizar el análisis, los *bugs* que eran descubiertos mediante las pruebas unitarias que se iban planteando a la par del desarrollo, eran resueltos en el momento y no se les relacionaba una incidencia, lo cual hizo imposible realizar un seguimiento de estos.

Bugs resueltos por *sprint*

- Medida: Número de *bugs* que fueron corregidos durante un *sprint*.
- Forma de medir: Contabilizar la cantidad de *bugs* marcados como completados por *sprint*.

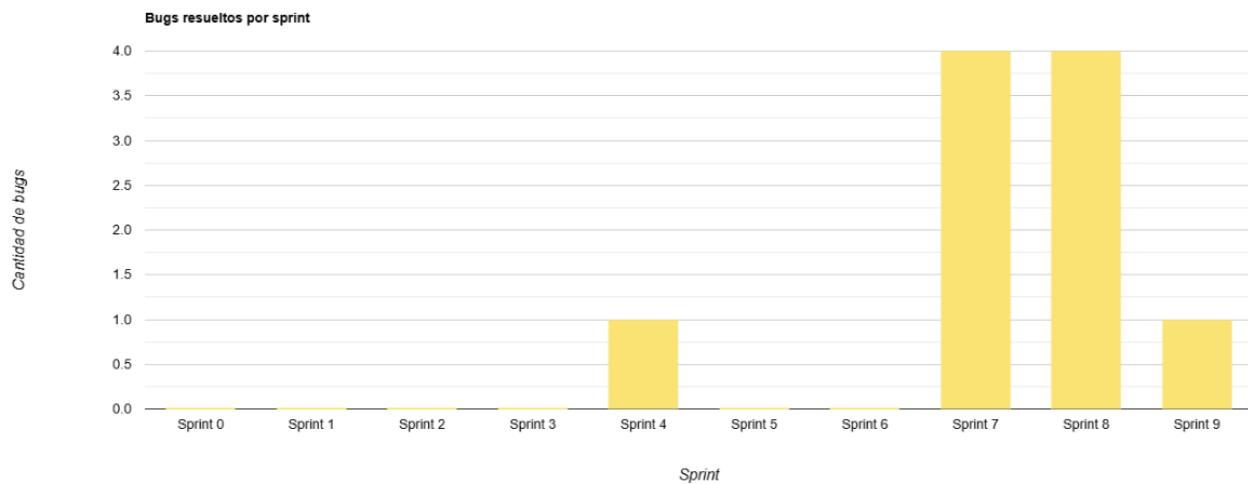


Ilustración 35 - Bugs resueltos por sprint

La métrica de *bugs* resueltos por *sprint* (Ilustración 35) muestra una clara correlación con la cantidad de *bugs* reportados. En los primeros *sprints* (0 a 2), no se corrigieron *bugs*, lo que es coherente con la ausencia de errores reportados en esas etapas tempranas del proyecto. A medida que el sistema avanzaba, en los *sprints* 3 y 5 se corrigieron algunos errores, pero en cantidades pequeñas, reflejando la baja complejidad del sistema en esos momentos.

El comportamiento cambia significativamente en los *sprints* finales (7 a 9), donde se observa un notable incremento en los *bugs* corregidos, alineado con el pico de errores reportados en esos mismos *sprints*. Aunque no se lograron resolver todos los *bugs*, el equipo respondió de manera bastante efectiva, resolviendo una buena parte de ellos, lo que refleja un esfuerzo por mantener el sistema lo más estable posible hacia las últimas etapas del proyecto.

En el *sprint* 9, aunque aún se corrigieron algunos *bugs*, la cantidad fue menor, lo que se explica en parte por la reducción de *bugs* reportados. Además, en este último *sprint*, el equipo priorizó la documentación para la entrega final, lo que pudo haber limitado la capacidad de resolver todos los errores pendientes.

Tiempo invertido en resolución de *bugs* vs desarrollo

- Medida: Porcentaje de tiempo dedicado a corregir *bugs* en comparación con el tiempo dedicado a desarrollar nuevas funcionalidades.
- Forma de medir: Contabilizar horas destinadas a resolución de *bugs* y horas destinadas al desarrollo de nuevas funcionalidades. Representar peso en porcentaje del tiempo dedicado a cada una dentro del tiempo total entre ambas.

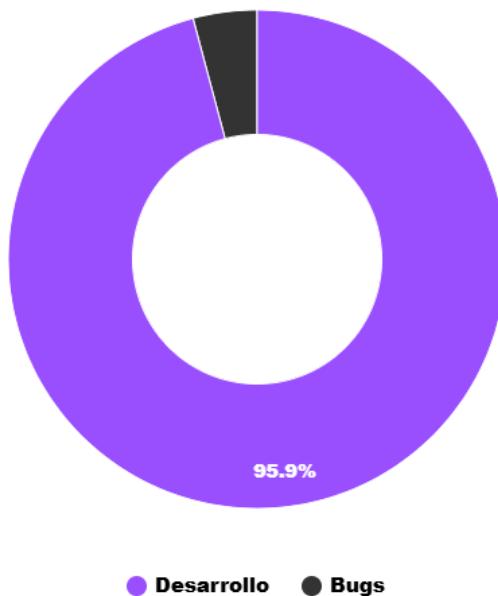


Ilustración 36 - Tiempo invertido en resolución de bugs vs desarrollo

El gráfico (Ilustración 36) muestra que, si sumamos el tiempo de desarrollo y el de resolución de bugs, la relación es la siguiente:

- El 95.9% del tiempo fue dedicado al desarrollo de nuevas funcionalidades.
- El 4.1% del tiempo se destinó a la resolución de *bugs*.

Esto indica que el equipo enfocó la mayor parte de sus esfuerzos en avanzar con nuevas características, lo que sugiere un proceso de desarrollo relativamente estable, con pocos problemas críticos a resolver.

El bajo porcentaje dedicado a la corrección de errores refleja una buena gestión de calidad del código y una baja incidencia de fallos significativos, permitiendo al equipo concentrarse en incrementar el valor del producto en lugar de dedicar demasiado tiempo a la corrección de fallos.

Cobertura de código de pruebas unitarias

- Medida: Porcentaje de líneas de código cubiertas por pruebas unitarias.
- Forma de medir: Mediciones arrojadas luego de los *tests* unitarios con herramienta elegida.

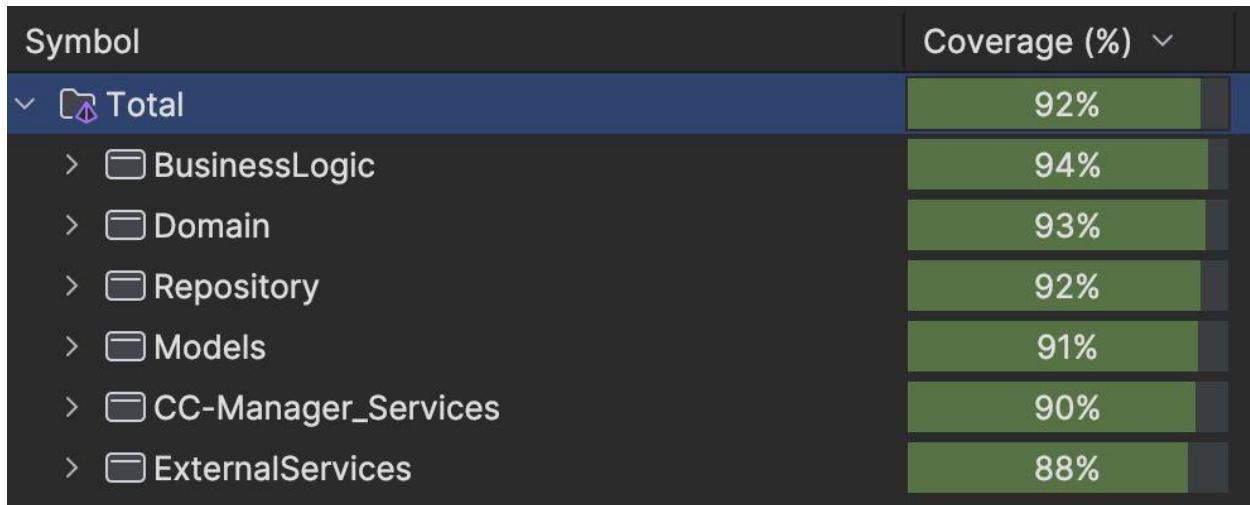


Ilustración 37 - Cobertura de testing

El gráfico (Ilustración 37) muestra que la cobertura total de pruebas unitarias alcanza un 92%, superando el objetivo establecido de mantener la cobertura por encima del 90%. Esto indica que el equipo ha logrado cubrir una parte significativa del código con pruebas unitarias, lo que refuerza la calidad y estabilidad del software.

7.4.5.2. Métricas de proceso

Distribución del esfuerzo del equipo según tipo de actividad

- Medida: Porcentaje del esfuerzo del equipo destinado a actividades como desarrollo, investigación, documentación, etc.
- Forma de medir: Contabilizar horas destinadas a cada una de estas actividades. Representar peso en porcentaje del tiempo dedicado a cada una dentro del tiempo total entre todas.

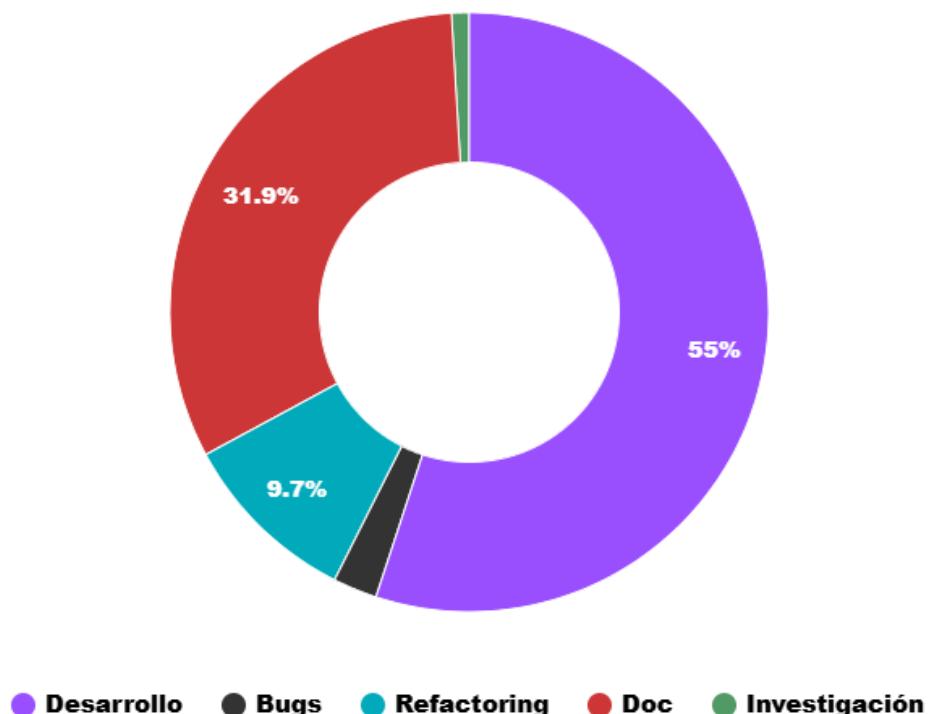


Ilustración 38 - Distribución del esfuerzo del equipo según tipo de actividad

El gráfico (Ilustración 38) muestra la distribución del esfuerzo del equipo en diversas actividades. La mayor parte del tiempo, un 55%, fue dedicada al desarrollo, lo que refleja que más la mitad del esfuerzo del equipo se centró en la creación de nuevas funcionalidades.

Un 31.9% se destinó a la documentación, lo que demuestra que esta actividad fue una prioridad para el equipo, incrementándose aún más el tiempo y la dedicación en este apartado durante las últimas semanas del proyecto.

Las otras actividades, como la resolución de *bugs*, la refactorización y la investigación, tuvieron menor peso en el tiempo total del proyecto. Esto refleja un enfoque principal en avanzar con el desarrollo y la documentación, manteniendo un balance adecuado en la corrección de errores y la mejora del código (refactorización).

Cabe aclarar, que este análisis solo enmarca el tiempo entre el *sprint 0* y el *sprint 9*, razón por la cual la investigación tiene tanto peso a pesar de haber contado con numerosas horas previo al inicio del primer *sprint*, como ya fue indicado anteriormente en el documento.

Velocidad del equipo

- Medida: Promedio de puntos de historia completados por *sprint*.
- Forma de medir: Promediar los puntos de historia completados en todos los *sprint* del proyecto.

Esta métrica ya fue discutida en esta [sección](#) del documento.

Tasa de deuda técnica (Desarrollo vs Refactorización)

- Medida: Porcentaje de tiempo dedicado a refactorizar código con respecto al tiempo invertido en desarrollar.
- Forma de medir: Contabilizar horas destinadas a refactorización de código y horas destinadas al desarrollo de nuevas funcionalidades. Representar peso en porcentaje del tiempo dedicado a cada una dentro del tiempo total entre ambas.

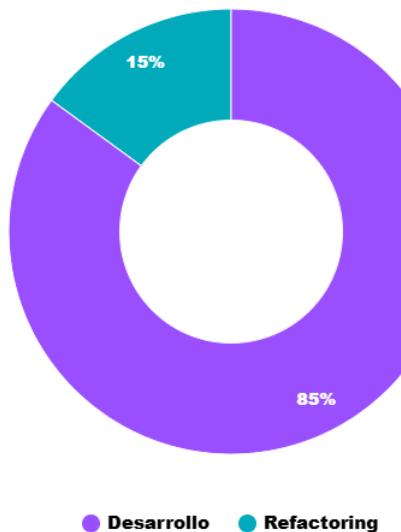


Ilustración 39 - Tasa de deuda técnica

El gráfico (Ilustración 39) refleja la tasa de deuda técnica, mostrando que un 85% del tiempo fue destinado al desarrollo de nuevas funcionalidades, mientras que un 15% se dedicó a la refactorización del código.

Este balance muestra que el equipo priorizó avanzar en el desarrollo del proyecto, dedicando la mayor parte del tiempo a la creación de nuevas características. Sin embargo, la refactorización, que representa casi el 15%, también recibió una atención considerable, lo que remarca que el

equipo hizo esfuerzos para mejorar la calidad del código y reducir la deuda técnica acumulada a lo largo del desarrollo.

Este enfoque balanceado contribuyó a garantizar un código más limpio y mantenible, lo que es esencial para evitar problemas futuros y mantener la calidad del *software* a largo plazo.

Uno de los objetivos que el equipo se había propuesto, era lograr dejar esta relación por debajo del 10%, por lo cual este objetivo no llegó a ser cumplido. En [esta sección](#), se comenta acerca de las razones aludidas.

Satisfacción del cliente

- Medida: Nivel de satisfacción del cliente con el producto entregado.
- Forma de medir: Encuestas de satisfacción al cliente al entregar el producto finalizado

Como ya ha sido discutido en este documento, el alcance de este proyecto es un *MVP*, por lo que el producto final no ha sido entregado al cliente aún. Esta encuesta será realizada una vez se realice dicha entrega.

En esta [sección](#), puede observar la encuesta planificada.

7.5. Conclusiones

En resumen, la gestión de calidad en este proyecto estuvo enfocada en asegurar tanto la calidad del producto como del proceso. Se implementaron diversas métricas para garantizar la satisfacción del cliente y mantener los estándares acordados. El equipo priorizó el desarrollo de nuevas funcionalidades, asegurando también una adecuada cobertura de pruebas unitarias y manteniendo el balance con actividades de refactorización y documentación.

El equipo alcanzó la meta de cobertura de pruebas unitarias, superando el 90%, lo que refuerza la estabilidad del sistema. Aunque no se logró cumplir con el objetivo de mantener la deuda técnica por debajo del 10%, se realizaron esfuerzos significativos para mejorar la calidad del código. Finalmente, se logró un enfoque equilibrado, con una atención adecuada a la documentación y la refactorización, asegurando la calidad del proyecto hacia su fase final.

8. Gestión de la configuración

En el transcurso de este capítulo, se ahondará sobre los detalles de los elementos y artefactos identificados por el equipo, así como también las herramientas escogidas para realizar el trabajo.

El objetivo de la gestión de la configuración del *software* es administrar los cambios que vayan surgiendo a lo largo de la vida del proyecto, gestionar el versionado de los componentes del sistema, así como también la preparación de este para futuras entregas externas del *software*.

8.1. Elementos de configuración de *software*

A lo largo de la vida del proyecto, se identificaron numerosos elementos que necesitaban ser almacenados y gestionados. No solo se hace referencia al código fuente, sino también a toda la documentación que fue desarrollándose para mantener un registro de las decisiones tomadas por el equipo, investigaciones realizadas y los hitos alcanzados.

Dentro de estos dos grandes conjuntos, se identificaron:

- *Software*:
 - Código fuente *frontend*.
 - Código fuente *backend*.
- Documentación:
 - Registros de reuniones internas del equipo.
 - Documentación académica.
 - Documentación de desarrollo.
 - Documentación de decisiones tomadas por el equipo.
 - Documentos de investigación.

8.2. Elección de herramientas

8.2.1. Versionado del *software* desarrollado

Como equipo, se determinó que la opción más adecuada era elegir Git como herramienta para el seguimiento, versionado y almacenamiento del *software* desarrollado.

Se pueden señalar varios factores que influyeron en esta decisión:

Utilización en la industria: Dentro de la industria del *software*, Git es una de las herramientas más utilizadas para estos fines. Este hecho le garantiza al equipo su confiabilidad y compatibilidad con las buenas prácticas de desarrollo estándares.

Experiencia del equipo: Todos los miembros del equipo cuentan con experiencia en el uso de esta herramienta, tanto a nivel profesional como académico, factor que facilita la implementación de su uso, además de un manejo eficiente de la herramienta.

Ambiente colaborativo: Dentro de este punto se encontraron dos grandes fortalezas. La primera, es que Git facilita que cada miembro trabaje de manera independiente y paralela en su propio entorno, lo cual facilita la división de las tareas. Adicionalmente, Git es una herramienta que permite colaborar en el desarrollo del *software*, permitiendo al equipo trabajar de manera conjunta de una manera muy eficiente.

Una vez elegida la herramienta para el control de versiones del *software*, fue momento de elegir la plataforma a utilizar para los servicios de repositorios.

Dada la vasta experiencia del equipo, la gran adopción a nivel de industria y academia y el hecho de poder obtener una licencia "PRO" de manera gratuita a través de la universidad, es que se optó por utilizar GitHub como proveedor de servicios de repositorios.

8.2.2. Documentación

Al inicio del proyecto, cuando llegó el momento de elegir dónde alojar la documentación que se desarrollaría durante los meses siguientes, el equipo definió encontrar una opción que satisfaga los siguientes puntos:

- Alojamiento en la nube
- Trabajo simultáneo
- Historial de versiones
- Facilidad de uso

Para esto, fueron tomadas en cuenta dos herramientas, Google Drive y Office (a través de Microsoft Teams).

Por ser la más conocida y utilizada por los integrantes del equipo, además de cumplir con los requerimientos, fue escogido Google Drive como herramienta para los elementos de documentación.

Fue creada una carpeta principal del proyecto, que dentro contenía múltiples subcarpetas asociadas a distintas tareas y etapas del proyecto. Por ejemplo, fueron almacenados todos los documentos generados para la presentación del proyecto dentro de una subcarpeta. También fueron creadas subcarpetas asociadas a cada capítulo de este documento, donde se almacenaban distintos recursos a ser utilizados para la confección de un documento final.

Dentro de la suite de Google Drive, fueron aprovechados las herramientas "Docs", para la creación de documentos de texto, así como también "Sheets" para la creación de documentación asociada a la gestión del riesgo.

Adicionalmente, para la creación de documentación específica, como diagramas, el equipo decidió utilizar "draw.io" como proveedor. También se hizo uso esporádico de boards de "miró".

Finalmente, para la creación de este documento final, se decidió utilizar Microsoft Word como editor de texto. Primeramente, fue utilizado "Docs", pero no contaba con la misma robustez que Word para la elaboración de documentación académica. Este documento generado, fue almacenado en el equipo de Microsoft Teams que fue creado con propósito de este proyecto.

8.3. Gestión del repositorio Git

Como equipo, adoptamos el estándar de GitFlow para la gestión de los repositorios de código. Esto fue muy importante para poder mantener un gran nivel de organización, así como también lograr una alta eficiencia colaborativa como equipo.

Dentro de este estándar, fueron establecidas dos ramas principales para el flujo del trabajo:

- Main (rama principal): rama donde se encuentra la versión estable y funcional del proyecto.
- Develop (rama de desarrollo): rama utilizada para sumar nuevos cambios y nuevas características al sistema.

A su vez, además de las ramas principales, fueron definidas categorías para la creación de otras ramas por parte de cualquier integrante del equipo.

- **Fix:** para la resolución de problemas surgidos y corregir errores previamente identificados, fueron creadas ramas con el sufijo *fix* para su identificación. El uso de estas ramas para estas tareas, aseguraban que la rama *main* se mantuviera confiable y estable.
- **Feature:** a la hora de desarrollar nuevas funcionalidades o agregarle nuevas características al proyecto, fueron creadas ramas con el sufijo *feature*. Estas siempre se desprenden de la rama *develop* y su principal característica es el aislamiento del desarrollo de estas nuevas funciones o características. Esto favorece la claridad a la hora del desarrollo de cada elemento.

Una vez finalizado el desarrollo en cualquiera de estas ramas, era necesario integrar estos cambios en la rama *develop*. Con este fin, el equipo decidió poner en práctica el uso de los *pull request*. Esta funcionalidad, permitía a los integrantes del equipo revisar y validar los cambios generados antes de que estos fueran incluidos en la rama *develop*. Una vez que dicho *pull request* se encontraba corregido y aprobado, era hora de integrarlo a la rama *develop*, lo que se conoce como

una operación de *merge*. Esta práctica permitió al equipo mantener un estándar de calidad además de un código coherente.

Los cambios de la rama *develop* serán integrados en la rama *main*, una vez que se considere que tiene los cambios suficientes y requeridos para la creación de una nueva versión. A la fecha de entrega de este documento, este proceso fue realizado una única vez, pero será efectuado múltiples veces en el futuro. El código contenido dentro de la rama *main* es el código de la versión que en un momento dado se encuentra en producción.

8.3.1.1. Reglas establecidas (Github actions)

Dentro de las reglas de integración entre ramas que ofrece Github, el equipo decidió hacer uso de tres de ellas:

- Sólo se pueden integrar los cambios a la rama *develop*, si el *pull request* fue aprobado por al menos uno de los integrantes del equipo.
- Solo se pueden integrar los cambios a la rama *develop*, si el 100% de los *test* asociados al nuevo desarrollo a integrar tienen como resultado "aprobado".
- Una vez que se ejecutaba la operación de *merge* en la rama *develop*, se eliminaba automáticamente la rama asociada a los nuevos cambios.

8.3.1.2. Integración con JIRA

Como ya fue mencionado anteriormente, JIRA fue la herramienta de gestión que el equipo utilizó a lo largo de todo el desarrollo. Esta herramienta cuenta con una integración con GitHub, lo cual permitía al equipo una sincronización entre una tarea establecida y una rama creada asociada a ella.

8.4. Conclusiones

La gestión de configuración resultó ser muy útil para el proyecto, proporcionando un control eficiente de las tareas y facilitando la organización en el versionado de los artefactos utilizados. Asimismo, fue una herramienta clave que le permitió al equipo recuperarse de errores, restaurando la última versión estable sin pérdida de datos.

Las herramientas seleccionadas para la documentación y la administración del *software* aseguraron una gestión constante, dinámica y bien organizada, lo que dejó al equipo satisfecho con los resultados obtenidos.

9. Conclusiones

9.1. Estado actual

Actualmente, el *MVP* del proyecto se encuentra en fase de refinamiento y pruebas exhaustivas. Aunque el módulo de análisis de costos mediante *machine learning* queda pendiente para después de la entrega final al cliente (por motivos ya explicados), el sistema ha demostrado ser funcional en sus características y en la mayoría de los requerimientos principales.

Las pruebas están permitiendo identificar posibles ajustes y mejoras antes de la entrega definitiva, lo que garantiza que el producto cumplirá con las expectativas y los requisitos del cliente.

9.2. Pasos a seguir a nivel de producto

Completar los requerimientos de alta prioridad pendientes

No quedaron requerimientos de alta prioridad pendientes para desarrollar. El *MVP* que presenta el equipo incluye todos los requisitos mínimos indispensables para ser considerado un *MVP*. Sin embargo, aunque los reportes e ingresos automatizados están listos, aún requieren algunos ajustes finales debido a que el ingreso no es recurrente en segundo plano, sino que debe ser iniciado a partir de dos fechas y un botón por el usuario, y se necesita avanzar un poco con la generación de más tipos de reportes.

Optimización del rendimiento

El equipo se centrará en revisar y optimizar el código, eliminando cualquier fragmento que pueda generar retrasos o demoras en el rendimiento del sistema. El objetivo es asegurar que el sistema funcione de manera óptima y fluida, sin interrupciones o tiempos de espera innecesarios, lo que garantizará una experiencia de usuario eficiente y confiable.

Pruebas finales exhaustivas

Tras la optimización del código, se llevarán a cabo pruebas finales exhaustivas. Estas pruebas cubrirán desde escenarios de uso real por medio del testing del cliente, hasta la verificación de la estabilidad del sistema bajo diferentes condiciones. Se pondrá especial énfasis en la usabilidad y en la detección de cualquier posible error que pudiera afectar la experiencia del usuario, asegurando que el sistema esté listo para su entrega en diciembre.

Soporte y plan de mantenimiento

Luego del *release* del 1 de diciembre, comienza oficialmente la etapa de soporte que anteriormente fue descrita.

Debido a que, en diciembre, integrantes del equipo se liberan las horas que le dedicaban al proyecto, las mismas pueden ser utilizadas para ofrecer soporte al cliente, ya sea por dudas/inconvenientes o *bugs* que se presenten.

El equipo estableció que el plan de mantenimiento del sistema se establezca por medio de dos fases bien diferenciadas. Durante la primera fase, se cubrirán las primeras dos temporadas completas como parte del acuerdo inicial con el cliente, sin costos adicionales para el mismo. Durante este período (llamémoslo garantía), el equipo estará completamente disponible para dar apoyo en el mantenimiento, y la implementación de mejoras. Una vez que concluya este ciclo, se procederá a una nueva negociación con el cliente, la cual que definirá aspectos clave como el precio y las dinámicas del mantenimiento, los integrantes del equipo responsables de llevar a cabo el mantenimiento, etc.

El plan incluye la posibilidad de recibir asesoría y asistencia técnica directamente en las instalaciones del cliente si es necesario, especialmente para situaciones complejas donde la presencia física facilite/agilice la resolución. Este tipo de soporte será liderado principalmente por el integrante del equipo Francisco, quien posee un gran conocimiento del sistema y de los procesos de la operativa del hotel.

En lo que corresponde a las actualizaciones del sistema, se contemplan dos tipos de *releases*. Inicialmente, durante las primeras tres versiones posteriores a la entrega del sistema, se realizarán *releases* cada dos semanas. Luego, la frecuencia de las versiones regulares será mensual, con la posibilidad de incluir versiones de emergencia en caso de situaciones críticas que afecten el funcionamiento del sistema. Las versiones de emergencia se lanzarán en el menor tiempo posible, garantizando que el sistema siga funcionando sin interrupciones graves.

El tratamiento de los problemas reportados se organizará de acuerdo con un nivel de gravedad. Para los problemas que sean catalogados de alta gravedad (que bloquen por completo el uso del sistema), el compromiso es responder en un plazo máximo de 24 horas desde el reporte inicial, buscando resolver y publicar una solución en el menor tiempo posible. En caso de problemas de gravedad media, la respuesta se dará en un plazo de 48 horas, momento en el que se evaluará junto con el cliente el tiempo adecuado para liberar una nueva versión que incluya la solución. Finalmente, para los problemas de baja gravedad, la aceptación del reporte se realizará dentro de las 72 horas, formando parte del siguiente *release* mensual si el reporte se efectúa al menos una semana antes del mismo.

Desarrollo de módulo de *machine learning*

Luego del *release* oficial en diciembre, el equipo se dispondrá a realizar una pausa (en lo que refiere a nuevos desarrollos), hasta comenzado el año 2025.

Una vez llegado el momento, se comenzará a planificar a alto nivel, los *sprints* que abordaran el desarrollo del módulo de *machine learning*, el cual sería el trabajo final del producto acordado con el cliente en los requerimientos.

9.3. Evolución y cumplimiento de los objetivos

9.3.1. Proyecto

9.3.1.1. Desarrollo de un *MVP*

KPI: Implementación del *MVP* del sistema (Sí/No)

Resultado: Se cumplió

Se logró desarrollar e implementar el sistema de gestión de inventario y costos que automatiza los procesos clave de registro y control de inventario de alimentos y bebidas del hotel.

Al momento de la entrega en gestión, se cuenta con un *MVP* que satisface las necesidades más básicas de la operativa del hotel para poder funcionar.

Al ser un *MVP*, no es la versión definitiva del sistema, ya que son cosas diferentes. La versión definitiva será publicada en el *release* de producción en diciembre.

9.3.1.2. Integración de tecnología avanzada

KPI: Integración de algoritmos de *machine learning* (Sí/No)

Resultado: No se cumplió

El equipo no logró cumplir con este objetivo. Básicamente, lo que sucedió es que hubo una priorización de los requerimientos, donde el módulo de análisis de costos a futuro por medio de *machine learning*, fue aplazado por el cliente, para tiempo después de la entrega, de manera que el equipo pueda centrar esfuerzos en los requerimientos más importantes.

Con esto, a pesar de que el equipo incumple con el objetivo, se gana tiempo.

Además, para que el módulo de *machine learning* funcione, necesita sí o sí que estén implementados los requerimientos anteriores, debido a que, si no, no se contaría con datos suficientes para lanzar predicciones.

9.3.1.3. Flexibilidad y escalabilidad del sistema

KPI: Capacidad del sistema para integrar módulos adicionales (Sí/No)

Resultado: Se cumplió

El sistema fue diseñado con una arquitectura modular que permite la fácil integración de nuevos módulos. El servidor no se acopla con ninguno de los dos clientes que se utilizan, y, además, gracias al uso de interfaces y polimorfismo en la *API*, resulta menos complicado aún extender al sistema, minimizando la necesidad de realizar modificaciones.

9.3.2. Producto

9.3.2.1. Completitud de requerimientos prioritarios/esenciales del producto

KPI: El 100% de las funcionalidades prioritarias/detalladas en el documento de requisitos debe estar completado y validado con éxito mediante pruebas funcionales internas antes de la entrega final. (Sí/No)

Resultado: Se cumplió

El objetivo se considera cumplido porque no quedaron requerimientos de alta prioridad pendientes. El MVP incluye todos los requisitos mínimos indispensables. Sin embargo, aunque los reportes e ingresos automatizados están listos, requieren ajustes adicionales, pero no afectan el cumplimiento de los requerimientos esenciales.

9.3.2.2. Garantizar la compatibilidad y adaptabilidad del sistema en múltiples plataformas

KPI: Compatibilidad en múltiples plataformas validada en pruebas (Sí/No)

Resultado: Se cumplió

El sistema fue probado en diferentes plataformas, incluyendo dispositivos móviles y de escritorio. Las pruebas en navegadores como Chrome en dispositivos móviles (Ilustración 40 e Ilustración 41) y de escritorio (Ilustración 42) confirmaron que la aplicación *web* es completamente responsive, adaptándose sin problemas a diferentes resoluciones de pantalla. Además, la aplicación Flutter fue compilada y probada en tablets Android (Ilustración 43) e iOS (Ilustración 44), garantizando el correcto funcionamiento en ambos sistemas operativos.

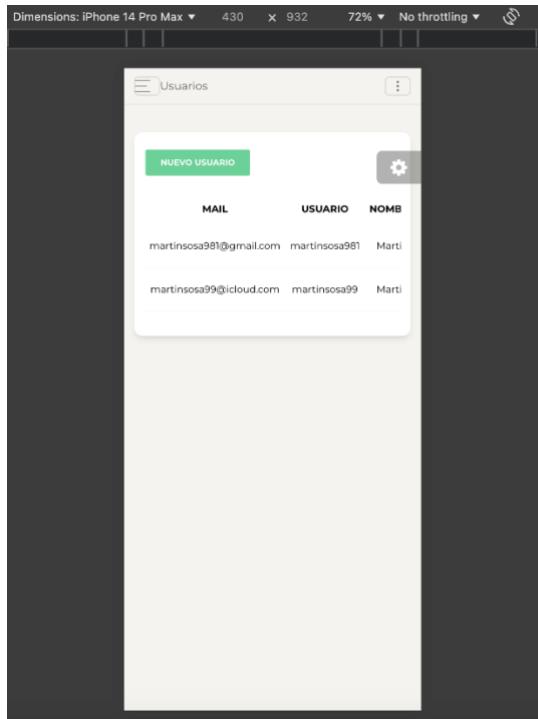


Ilustración 40 - Aplicación web en celular (modo vertical)

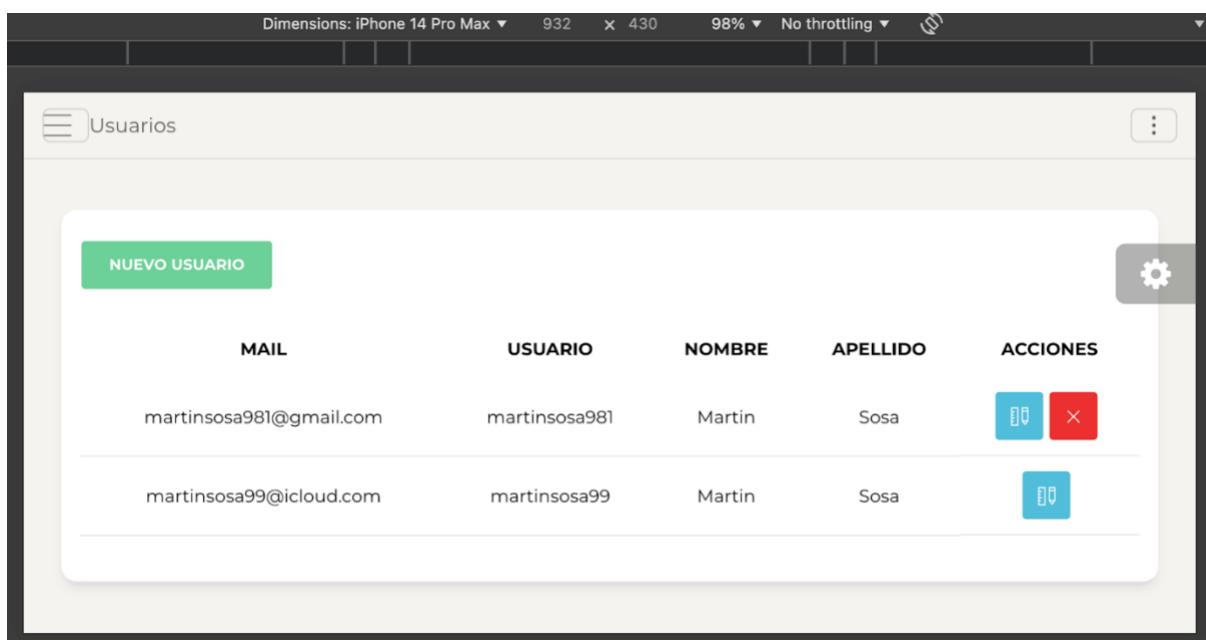


Ilustración 41 - Aplicación web en celular (modo horizontal)

The screenshot shows a web application interface titled "CHEFCHECK". On the left, there is a sidebar with the following menu items:

- PERSONAL
- MENU
- USUARIOS
- PROVEEDORES
- TIPOS DE PRODUCTO
- PRODUCTOS
- FACTURAS
- HABITACIONES

The main content area is titled "Usuarios" and displays a table with one row of data:

MAIL	USUARIO	NOMBRE	APELLIDO	ACCIONES
franciscoecabanillas@gmail.com	fcabanillas	Francisco	Cabanillas	

Ilustración 42 - Aplicación web en computadora

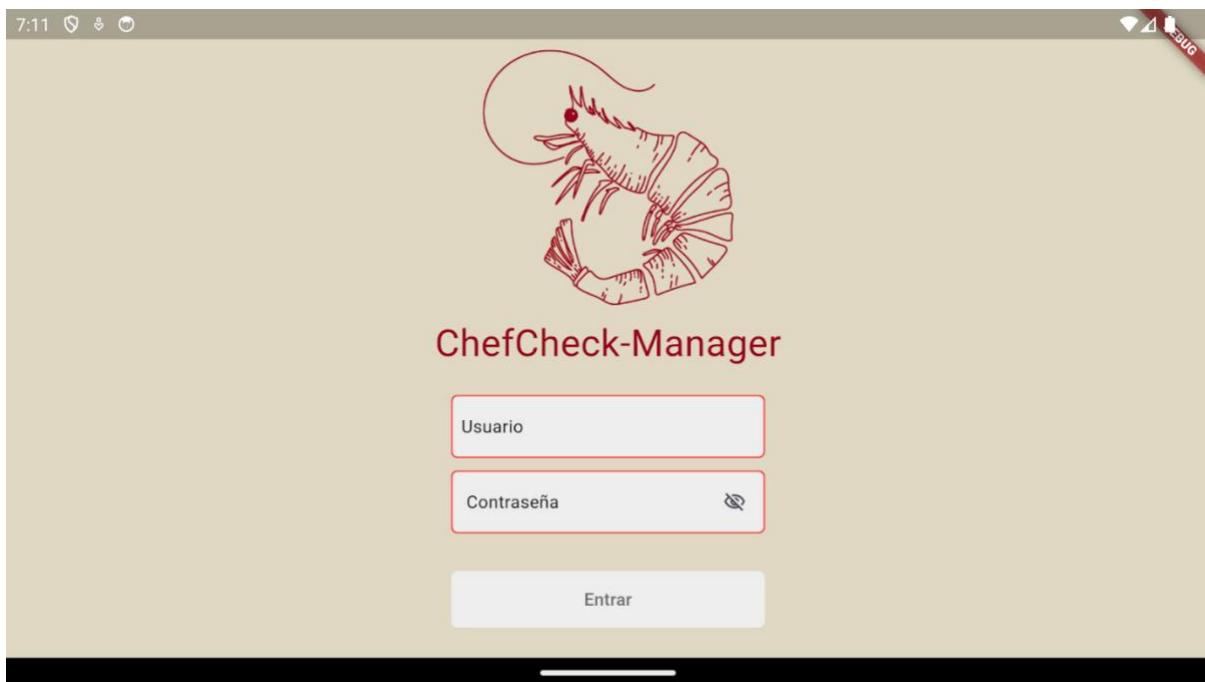


Ilustración 43 - Aplicación móvil en Android

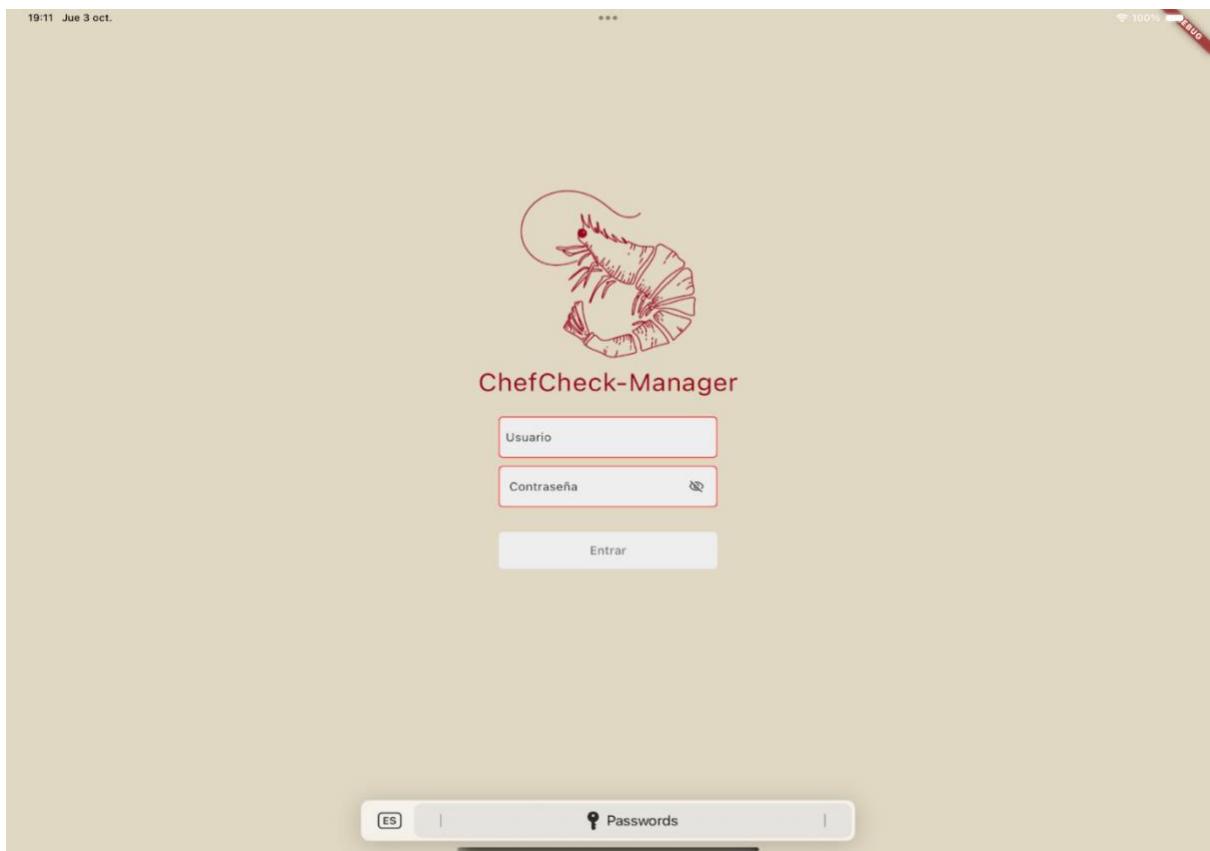


Ilustración 44 - Aplicación móvil en iOS

9.3.2.3. Calidad y profesionalismo del producto

KPI: Nivel de satisfacción del usuario con la interfaz y el uso (Escala 1-5)

Resultado: No se cumplió aún

El cumplimiento o no de dicho objetivo, únicamente podrá ser evaluado durante la post entrega del presente documento, debido a que las dos sesiones restantes de verificación de requerimientos ocurrirán durante los *sprints* 10 (desde 17/10/2024 hasta 4/11/2024) y 11 (desde 4/11/2024 hasta 18/11/2024).

La usabilidad con el cliente será evaluada por medio de una encuesta realizada a los usuarios del sistema (propietario, gerente operacional, encargada del comedor, chef, cocineros, encargado del stock, encargado de la barra y barman), una vez hayan probado el sistema.

Las preguntas que se realizarán (ya están diseñadas), y serán enviadas por medio de un Google Forms. Como la usabilidad es uno de los requerimientos no funcionales, se puede visualizar en este anexo [RNF-5 - Usabilidad](#).

9.3.3. Académicos

9.3.3.1. Aprendizaje y aplicación de nuevas tecnologías

KPI: Adopción de tecnologías nuevas y desafiantes (o escasamente utilizadas por el equipo) como Python (Sí/No)

Resultado: No se cumplió

El equipo no logró cumplir con este objetivo. Básicamente, lo que sucedió es que hubo una priorización de los requerimientos, donde el módulo de análisis de costos a futuro por medio de *machine learning*, fue aplazado por el cliente, para tiempo después de la entrega, de manera que el equipo pueda centrar esfuerzos en los requerimientos más importantes.

Con esto, a pesar de que el equipo incumple con el objetivo, se gana tiempo.

Además, para que el módulo de *machine learning* funcione, necesita sí o sí que estén implementados los requerimientos anteriores, debido a que, si no, no se contaría con datos suficientes para lanzar predicciones.

9.3.3.2. Aplicación práctica de conocimientos académicos

KPI: Cantidad de técnicas académicas aplicadas en el proyecto

Resultado: Se cumplió, con al menos 4 técnicas aplicadas

Se aplicaron múltiples técnicas aprendidas a lo largo de la carrera, como ser: diseño de la arquitectura del *software*, modelado de bases de datos, gestión de proyectos con la metodología adaptada Scrum, pruebas automatizadas, etc. Esto permitió crear un proyecto sólido y funcional, combinando lo académico con la práctica profesional.

El equipo muestra conformidad y comodidad respecto a la aplicación de las técnicas aprendidas durante los cuatro años de carrera.

9.3.4. Equipo

9.3.4.1. Compromiso con el equipo

KPI: El 100% de los integrantes acudió a todas las reuniones clave (Scrum *daily, planning, review* y *retrospective*). (Sí/No)

Resultado: Se cumplió, todos los integrantes acudieron siempre a las reuniones.

El equipo siempre tuvo la libertad de cambiar de fecha las reuniones realizadas, para que justamente puedan asistir todos los integrantes. Eso demuestra la flexibilidad y compromiso del equipo con el proyecto. Pese a que inicialmente el día fijado para las reuniones entre el equipo (sin la tutora) eran los jueves, si alguno no podía se cambiaba de fecha. Lo mismo sucedía para las Scrum *planning* y *review* cada dos semanas (lunes por lo general). Las reuniones con la tutora Helena Garbarino fueron fijadas para todos los lunes a las nueve de la mañana. Como se estableció un horario fijo, sucedió en alguna ocasión que algún integrante no pudiese concurrir. No fue algo que afecte al equipo ya que siempre había al menos dos de los tres integrantes del equipo presentes en las reuniones con la tutora. Esta situación aconteció en únicamente dos de las reuniones con la tutora, ya que, durante el resto, estuvo la totalidad del equipo presente.

9.3.4.2. Creación de un emprendimiento tecnológico

KPI: Viabilidad del proyecto como emprendimiento (Sí/No)

Resultado: Se cumplió, el proyecto es económicamente viable, y visto como una oportunidad de negocio.

Durante el desarrollo del sistema, se evaluó su viabilidad comercial y se concluyó que la solución tiene potencial para expandirse a otros hoteles y negocios de la industria hotelera. El equipo está considerando convertir este proyecto en un emprendimiento a futuro, aprovechando las características flexibles y escalables del sistema.

En principio, *ChefCheck-Manager* será adquirido mediante un acuerdo (cuyos detalles preliminares se encuentran en la documentación de la cesión de derechos) con el cliente Hotel Tío Tom. En conclusión, al haber un rédito económico, el objetivo se cumple.

9.3.4.3. Disfrutar y aprender durante el desarrollo del proyecto

KPI: Nivel de satisfacción del equipo con el proceso de desarrollo (Escala 1-5)

Resultado: Se cumplió 4.8/5

El equipo reportó un alto nivel de satisfacción con el proceso de desarrollo, con una puntuación promedio de 4.8 sobre 5, lo cual el equipo interpreta que el objetivo fue cumplido.

El proyecto fue visto no solo como un desafío técnico, sino también como una oportunidad para crecer tanto profesional como personalmente, disfrutando cada etapa del proceso.

Al tratarse de un proyecto no individual, sino en equipo, el desafío más grande fue el hecho de congeniar entre los tres integrantes.

Para evaluar esto, los integrantes completaron una encuesta anónima, donde se realizaron las siguientes preguntas, evaluando con una puntuación de 1 a 5. El detalle se puede visualizar en el anexo [Objetivo "Disfrutar y aprender durante el desarrollo del proyecto"](#).

9.4. Lecciones aprendidas

A lo largo de este proyecto, el equipo ha enfrentado numerosos desafíos que le ha permitido aprender y crecer tanto a nivel profesional como personal. Desde el inicio, uno de los mayores retos fue trabajar sin la dirección constante de un profesor (debido a que el proyecto final, justamente es para que el equipo ponga en práctica todo lo aprendido durante la carrera).

Este proyecto le obligó al equipo a salir de su zona de confort, asumiendo el control total de las decisiones. Fue el primer proyecto profesional sin un superior o jefe a cargo, por lo cual, les llevó a tomar responsabilidad total sobre el desarrollo, gestión y ejecución de este.

El manejo del tiempo y el cumplimiento de los plazos fueron aspectos clave, ya que el equipo tuvo un compromiso firme con el cliente dentro de un tiempo acotado. Lograr equilibrar el tiempo demandado del proyecto con las responsabilidades laborales de cada integrante fue uno de los mayores desafíos. Aprender a gestionar el tiempo libre de cada uno, les permitió mejorar en términos de organización y disciplina personal.

En cuanto a la desviación de las estimaciones, aunque no fueron completamente precisas y fue la primera vez que el equipo asumió un rol protagónico en la toma de decisiones sobre las mismas, esto no impidió que se lograra crear el *MVP* a tiempo para la entrega de este documento. Igualmente, el desarrollo continúa, para seguir fortaleciendo el *MVP* con el que se cuenta a fecha de hoy.

Además, el proyecto llevó a que se tuviera que trabajar con tecnologías y herramientas con las que algunos miembros del equipo no estaban familiarizados, lo que nos sacó de nuestra zona de confort. Esta experiencia les enseñó a adaptarse rápidamente, a investigar de manera individual, lo cual fue un aprendizaje valioso que fortaleció las habilidades técnicas y la confianza para enfrentar desafíos tecnológicos en el futuro como equipo.

El trabajo en equipo fue un pilar fundamental en el éxito del proyecto. La colaboración y cooperación entre los miembros del equipo no solo fueron clave para resolver problemas, sino que también se vieron reflejadas en cada reunión que se realizó, ya fueran las daylies de Scrum, las reuniones de planificación o las revisiones de *sprint*.

Cada integrante aportó mucho mediante sus perspectivas y habilidades, lo que permitió que el equipo se mantuviera alineado y enfocado en los objetivos. La transparencia y el constante apoyo entre el equipo fueron elementos cruciales para superar los momentos más difíciles.

El apoyo de la tutora Helena Garbarino, también fue fundamental. A lo largo del proyecto, sus sugerencias ayudaron a mantener el rumbo adecuado. Su experiencia permitió ajustar y mejorar el proyecto en momentos clave, brindando la confianza necesaria para tomar decisiones importantes y poder seguir adelante con el proyecto.

Esta experiencia le ha enseñado al equipo no solo a enfrentar desafíos técnicos, sino también a gestionar el tiempo y a trabajar en equipo bajo presión a contrarreloj. El equipo aprendió a apoyarse mutuamente y a tomar decisiones clave. Salir de la zona de confort y enfrentar nuevas tecnologías les dio una visión más amplia y profunda de las propias capacidades del equipo, preparándolos mejor para futuros proyectos profesionales.

10. Referencias bibliográficas

- [1] Microsoft, « Quickstart: Create an Azure Database for PostgreSQL - Flexible Server instance in the Azure portal ». [En línea]. Available: <https://learn.microsoft.com/en-us/azure/postgresql/flexible-server/quickstart-create-server-portal>. [Último Acceso: 04 de Octubre 2024]
- [2] Microsoft, «App Service». [En línea]. Available: <https://azure.microsoft.com/es-es/products/app-service>. [Último Acceso: 04 de Octubre 2024]
- [3] Microsoft, «Static Web Apps». [En línea]. Available: <https://azure.microsoft.com/en-us/products/app-service/static>. [Último Acceso: 04 de Octubre 2024]
- [4] Lucidchart, «Tutorial de diagrama de secuencia UML» [En línea]. Available: <https://www.lucidchart.com/pages/es/diagrama-de-secuencia>. [Último Acceso: 27 de Setiembre 2024]
- [5] Creately, «Tutorial del diagrama de secuencia: Guía completa con ejemplos» Octubre 2022. [En línea]. Available: <https://creately.com/blog/es/diagramas/tutorial-del-diagrama-de-secuencia/>. [Último Acceso: 27 de Setiembre 2024]
- [6] ITDO, «¿Qué es BDD (Behavior Driven Development)?» Julio 2019. [En línea]. Available: <https://www.itdo.com/blog/que-es-bdd-behavior-driven-development/>. [Último Acceso: 14 de Mayo 2024]
- [7] Immune Institute, «5 Beneficios de trabajar con metodología SCRUM» Febrero 2023. [En línea]. Available: <https://immune.institute/blog/metodologia-scrum-caracteristicas/>. [Último Acceso: 16 de Mayo 2024]
- [8] Microsoft, «Introducción a .NET» Febrero 2024. [En línea]. Available: <https://learn.microsoft.com/es-es/dotnet/core/introduction>. [Último Acceso: 28 de Mayo 2024]
- [9] OpenWebinars, «Frameworks Java para un desarrollo eficiente,» Noviembre 2023. [En línea]. Available: <https://openwebinars.net/blog/frameworks-java-para-un-desarrollo-eficiente/>. [Último Acceso: 28 de Mayo 2024]
- [10] Microsoft, «Entity Framework Core» Agosto 2023. [En línea]. Available: <https://learn.microsoft.com/es-es/ef/core/>. [Último Acceso: 28 de Mayo 2024]
- [11] Dreams, «¿Qué es un ORM?». [En línea]. Available: <https://www.dreams.es/transformacion-digital/desarrolladores-paginas-web/que-es-un-orm>. [Último Acceso: 28 de Mayo 2024]

[12] UI From Mars, «10 reglas heurísticas de Nielsen y cómo aplicarlas» [En línea]. Available: <https://www.uifrommars.com/10-reglas-heuristicas-como-aplicarlas/>. [Último Acceso: 10 de Octubre 2024]

[13] Scrum Manager BoK, «INVEST» Diciembre 2023. [En línea]. Available: <https://www.scrummanager.com/bok/index.php/INVEST>. [Último Acceso: 20 de Octubre 2024].

[14] Curso Dirección de Proyectos, «Ciclos de vida iterativo e incremental, ¿Qué son?» Julio 2018. [En línea]. Available: <https://www.cursodireccionproyectos.com/2018/07/ciclos-de-vida-iterativo-e-incremental-que-son/>. [Último Acceso: 16 de Mayo 2024]

[15] Ingeniería de Software TDEA, «Ciclo de vida evolutivo» [En línea]. Available: <https://ingenieriadesoftwaretdea.weebly.com/ciclo-de-vida-evolutivo.html>. [Último Acceso: 16 de Mayo 2024]

[16] RESTful API, «What is REST?» Diciembre 2023 [En línea]. Available: <https://restfulAPI.net/>. [Último acceso: 21 de Octubre 2024].

[17] Google Github, «Google JavaScript Style Guide» [En línea]. Available: <https://google.github.io/styleguide/jsguide.html>. [Último acceso: 21 de Octubre 2024].

[18] Google Github, «TypeScript Style Guide» [En línea]. Available: <https://google.github.io/styleguide/jsguide.html>. [Último acceso: 21 de Octubre 2024].

[19] Github, «Style guide for Flutter repo» [En línea]. Available: <https://github.com/Flutter/Flutter/wiki/Style-guide-for-Flutter-repo/5c279fe8d7b52d31ebcf05385f69c636a6b8e676>. [Último acceso: 21 de Octubre 2024].

[20] Microsoft, «Common C# code conventions» Agosto 2023. [En línea]. Available: <https://learn.microsoft.com/en-us/dotnet/csharp/fundamentals/coding-style/coding-conventions>. [Último acceso: 21 de Octubre 2024].

11. ANEXO

11.1. Marco metodológico

11.1.1. Ventajas de Scrum

Scrum ofrece varias ventajas, como mejorar la satisfacción del cliente al permitir la entrega rápida y frecuente de productos de valor, facilitando la incorporación de *feedback*. Además, reduce los costes mediante la optimización del trabajo y la eliminación de tareas innecesarias. Scrum también promueve equipos más productivos y motivados, al proporcionar un enfoque colaborativo y garantizar un seguimiento continuo del progreso, lo que es ideal para gestionar proyectos complejos en entornos cambiantes [7].

11.2. Ingeniería de requerimientos

11.2.1. I.N.V.E.S.T

I.N.V.E.S.T es una técnica que se utiliza en el desarrollo ágil para asegurar la calidad de las *user stories*. Fue propuesto por Bill Wake en 2003 y consiste en una serie de características esenciales que deben cumplir estas historias para facilitar su gestión y asegurar que sean útiles en la planificación y ejecución de proyectos. Estas características son: que la historia sea independiente, para poder implementarse sin depender de otras. Negociable, para permitir ajustar detalles en conversaciones con los usuarios. Valiosa, al aportar un beneficio claro para el cliente. Estimable, para que el equipo pueda calcular el esfuerzo necesario. Pequeña, para que pueda completarse en un corto periodo. Comprobable, para verificar su correcta implementación mediante pruebas claras. Aplicar este enfoque mejora la comunicación y la planificación en los equipos ágiles. [13].

11.3. Arquitectura de la solución

11.3.1. Tecnologías

11.3.1.1. .NET

.NET es una plataforma de desarrollo gratuita, de código abierto y multiplataforma, diseñada para crear una amplia variedad de aplicaciones, desde web y móviles hasta juegos y sistemas de *internet of the things*. Se basa en un entorno de ejecución de alto rendimiento y admite varios lenguajes de programación, siendo C# el más popular. Con una arquitectura centrada en la productividad, seguridad y rendimiento, .NET ofrece características como la administración automática de memoria y la programación asincrónica. Además, su ecosistema incluye herramientas y bibliotecas optimizadas para facilitar el desarrollo en múltiples sistemas operativos y arquitecturas [8].

11.3.1.2. Java Spring

Java Spring es un framework que facilita el desarrollo de aplicaciones empresariales en Java, proporcionando un entorno modular y flexible. Su principal característica es la inversión de control, que permite gestionar las dependencias entre componentes de manera eficiente, promoviendo la reutilización de código y la modularidad. Spring también destaca por su capacidad de integración con otras tecnologías y *frameworks*, y ofrece herramientas avanzadas para la seguridad, como autenticación y autorización. Su extensión, Spring Boot, simplifica aún más el desarrollo al ofrecer configuraciones predeterminadas para agilizar la creación de aplicaciones empresariales, servicios web y microservicios [9].

11.3.1.3. Entity Framework Core

Entity Framework Core (EF Core) es una herramienta de acceso a datos multiplataforma y de código abierto que permite a los desarrolladores de .NET trabajar con bases de datos de manera más eficiente. Funciona como un mapeador relacional de objetos, lo que significa que permite manipular bases de datos usando objetos del lenguaje C# en lugar de escribir consultas SQL manualmente. EF Core es compatible con varios sistemas de bases de datos y facilita la gestión de las interacciones con estas mediante consultas, guardado de datos y migraciones automáticas, lo que simplifica la evolución del esquema de la base de datos a lo largo del tiempo [10].

11.3.1.4. ORM

Un ORM (*Object Relational Mapping*) es una herramienta que facilita la interacción entre una aplicación y una base de datos al convertir los objetos del código en un formato adecuado para ser almacenados en la base de datos. Esto elimina la necesidad de escribir manualmente consultas SQL para realizar operaciones como crear, leer, actualizar o eliminar datos (CRUD). Además, el ORM gestiona automáticamente las actualizaciones en las tablas y permite cambiar el motor de la base de datos sin modificar el código, simplificando el mantenimiento y mejorando la eficiencia del desarrollo [11].

11.3.1.5. Azure App Service

Azure App Service es una plataforma en la nube completamente gestionada que permite desarrollar, desplegar y escalar aplicaciones *web*, *APIs* y aplicaciones móviles de forma sencilla y eficiente. Proporciona un entorno seguro y escalable para ejecutar aplicaciones, eliminando la necesidad de gestionar la infraestructura subyacente, como servidores o平衡adores de carga. Con Azure App Service, el equipo puede enfocarse en el código y las funcionalidades de sus aplicaciones, mientras que Azure se encarga del mantenimiento y escalado automático de los recursos. Además, ofrece características avanzadas como integración y entrega continuas (CI/CD), autenticación y autorización integradas, monitoreo detallado del rendimiento, y soporte para múltiples lenguajes de programación, como .NET, Java, Node.js, Python, etc. Esto lo convierte en

una solución ideal para empresas que desean acelerar el tiempo de desarrollo y asegurar la alta disponibilidad de sus aplicaciones en producción [2].

11.3.1.6. Azure Static Web Apps

Azure Static Web Apps es un servicio en la nube de Microsoft que permite el desarrollo y despliegue de aplicaciones *web* estáticas de manera rápida y sencilla. Estas aplicaciones están compuestas por archivos HTML, CSS, JavaScript y otros recursos estáticos que se alojan y distribuyen a nivel global a través de la red de entrega de contenido (CDN) de Azure, lo que garantiza un rendimiento rápido y eficiente. El servicio incluye una integración automática con GitHub y Azure *DevOps* para manejar el flujo de trabajo de CI/CD, lo que significa que cada vez que se actualiza el código en el repositorio, Azure compila y despliega la aplicación automáticamente. Además, Azure Static Web Apps permite la configuración de dominios personalizados, certificados SSL automáticos y la implementación de funciones de *backend* mediante Azure Functions, lo que hace posible manejar solicitudes dinámicas. Este servicio es ideal para desarrolladores que buscan una solución simplificada para aplicaciones *web* ligeras, blogs, portafolios, sitios corporativos, entre otros [3].

11.3.1.7. Azure Functions

Azure Functions es un servicio de computación de Microsoft Azure que permite ejecutar código en respuesta a eventos sin necesidad de contar con un servidor. El servicio escala automáticamente según la demanda y cobra solo por el tiempo de ejecución del código, lo que lo hace ideal para tareas automatizadas, como procesamiento de datos, o desencadenar acciones basadas en eventos.

11.3.1.8. Azure Load Testing

Azure Load Testing es un servicio en la nube de Microsoft Azure que permite realizar simulaciones de cargas de trabajo en aplicaciones *web*, con el objetivo de medir su rendimiento y capacidad bajo ciertas condiciones. La herramienta ayuda a identificar cuellos de botella, medir tiempos de respuesta y verificar si las aplicaciones pueden manejar una cantidad creciente de tráfico sin degradarse.

11.3.2. Evolución de los requerimientos no funcionales

11.3.2.1. RNF-1 – Seguridad

El sistema garantiza la seguridad de los datos sensibles mediante el uso de protocolos como HTTPS y la tokenización JWT. Aunque, al momento de la entrega de este documento, aún no se ha adquirido un certificado HTTPS para el entorno de producción, está prevista su implementación en las etapas finales previo a la salida a producción. La *API* implementa JWT para gestionar las sesiones de usuario, asegurando que cada usuario esté autenticado y que los tokens sean validados

en cada solicitud. Además, los datos sensibles en la base de datos, como contraseñas, son cifrados para asegurar su confidencialidad.

11.3.2.2. RNF-2 – Disponibilidad

Para garantizar una disponibilidad mínima del 99% durante la temporada alta, el equipo utilizará la herramienta UptimeRobot (Ilustración 45), la cual se encargará de monitorear el sistema durante el primer mes posterior al lanzamiento. Esta herramienta proporcionará estadísticas que permitirán evaluar si el sistema cumple con los tiempos de disponibilidad requeridos. A partir de los datos recolectados, se tomarán las decisiones necesarias para ajustar la infraestructura en caso de ser necesario.

A priori, el equipo considera que no van a existir problemas de disponibilidad. Primeramente, porque se confía en la infraestructura de Azure de la empresa Microsoft. Segundo, debido a que el sistema en la mayoría del tiempo no será utilizado por más de dos usuarios en simultáneo. Aun así, las mediciones de disponibilidad se evaluarán.

Este análisis se realizará a medida que el sistema entre en producción (a partir del 1 de diciembre de 2024).

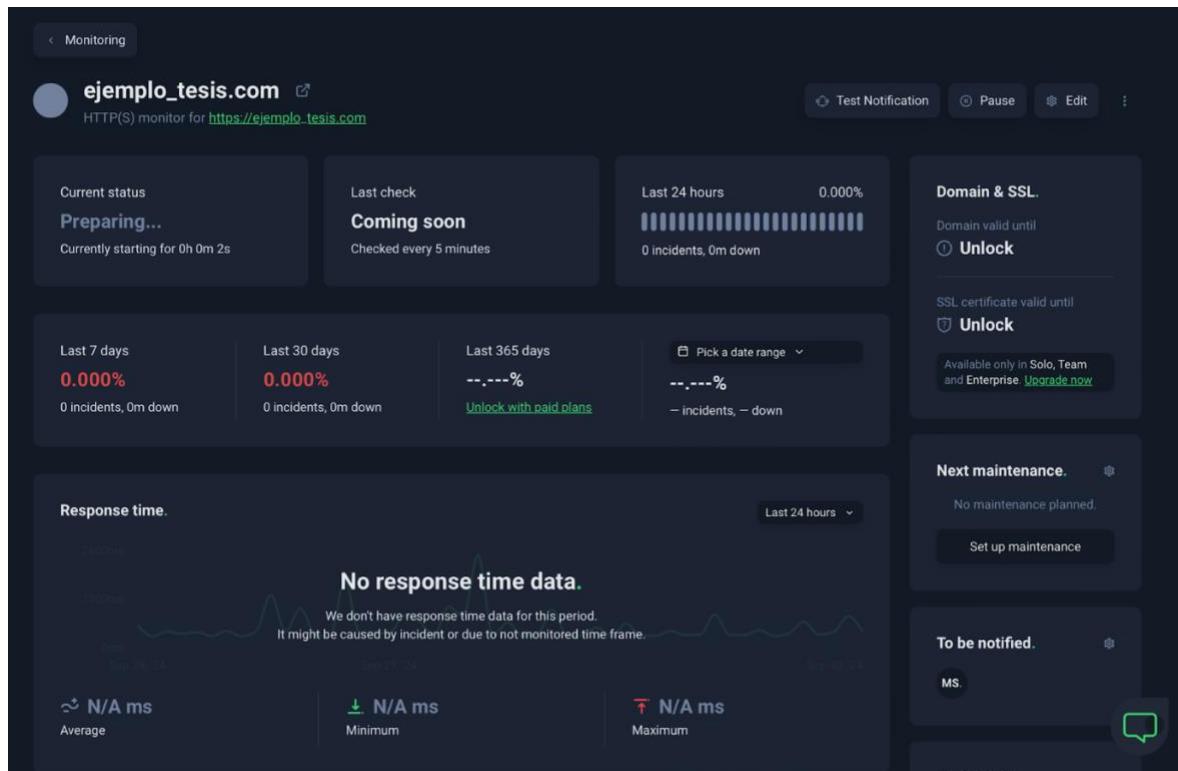


Ilustración 45 - Ejemplo de cómo se visualiza la interfaz gráfica de la herramienta UptimeRobot

11.3.2.3. RNF-3 – Escalabilidad

El sistema ha sido diseñado para ser escalable horizontalmente, lo que permitirá añadir más recursos de manera dinámica durante la temporada alta o en momentos de alta demanda. La infraestructura que actualmente se piensa utilizar, soporta la carga de trabajo que necesita el hotel. Azure ofrece la capacidad de escalar con facilidad.

Una de las grandes razones por las cuales se eligió Azure, es debido a la enorme cantidad de servicios que tiene, siendo uno de ellos la facilidad para escalar con facilidad.

11.3.2.4. RNF-4 – Rendimiento

El rendimiento del sistema será analizado una vez que quede alojado todo el sistema en Azure. Para validar este cumplimiento, se realizarán pruebas utilizando Azure Load Testing (en el anexo [Azure Load Testing](#) se da contexto acerca de lo que es esta herramienta).

Estas pruebas se deben llevar a cabo directamente en el entorno de producción en Azure, ya que realizar pruebas en entornos locales no ofrece resultados fiables.

Al igual que en el caso del requerimiento de disponibilidad, el rendimiento se seguirá evaluando a medida que transcurre la temporada del hotel.

11.3.2.5. RNF-5 – Usabilidad

El equipo realizó un análisis interno en cuanto a las heurísticas de Nielsen (las cuales se pueden observar en el anexo [Cumplimiento de las heurísticas de Nielsen](#)), donde se fue evaluando una a una, si la aplicación web y móvil cumplen con las mismas. Los resultados fueron favorables.

En lo que corresponde a usabilidad desde el punto de la interacción del cliente con el producto, el equipo tiene pensado realizar nuevas sesiones de verificación de los requerimientos durante los *sprints* 10 (desde 17/10/2024 hasta 4/11/2024) y 11 (desde 4/11/2024 hasta 18/11/2024), posterior a la entrega del documento en gestión.

La usabilidad con el cliente será evaluada por medio de una encuesta realizada a los usuarios del sistema (propietario, gerente operacional, encargada del comedor, chef, cocineros, encargado del stock, encargado de la barra y barman), una vez hayan probado el sistema.

Las preguntas que se realizarán ya están diseñadas (Ilustración 46, Ilustración 47, Ilustración 48, Ilustración 49, Ilustración 50, Ilustración 51, Ilustración 52, Ilustración 53 e Ilustración 54) y serán enviadas por medio de un Google Forms.

¿Cual de las siguientes aplicaciones utilizará en su día a día?

- Portal Web
- Aplicación mobile en la tablet

Ilustración 46 - Pregunta 1 de encuesta de usabilidad para el cliente

El sistema es fácil de usar y entender sin necesidad de asistencia técnica.



Ilustración 47 - Pregunta 2 de encuesta de usabilidad para el cliente

Es fácil encontrar las funciones y herramientas que necesito en el sistema.



Ilustración 48 - Pregunta 3 de encuesta de usabilidad para el cliente

El sistema responde rápidamente a mis acciones sin tiempos de espera prolongados.



Ilustración 49 - Pregunta 4 de encuesta de usabilidad para el cliente

La información presentada en el sistema es clara y comprensible.



Ilustración 50 - Pregunta 5 de encuesta de usabilidad para el cliente

Aprender a usar el sistema fue fácil y no requirió demasiado tiempo.



Ilustración 51 - Pregunta 6 de encuesta de usabilidad para el cliente

Los íconos, botones y gráficos del sistema son claros y me ayudan a entender mejor su funcionamiento.



Ilustración 52 - Pregunta 7 de encuesta de usabilidad para el cliente

Cuando cometo un error, el sistema me ofrece instrucciones claras para corregirlo.



Ilustración 53 - Pregunta 8 de encuesta de usabilidad para el cliente

Considero que sería útil tener un menú FAQ o una sección de ayuda para resolver dudas comunes sobre el uso del sistema.

1. Si
2. No

Ilustración 54 - Pregunta 9 de encuesta de usabilidad para el cliente

11.3.2.6. RNF-6 – Compatibilidad

Tanto el panel *web* como la aplicación móvil han sido probados en diferentes dispositivos y navegadores. Para demostrar la compatibilidad del sistema, se han generado capturas de pantalla que muestran el funcionamiento de la *web* en Google Chrome, tanto en vertical como horizontal (Ilustración 55, Ilustración 56 e Ilustración 57), y de la aplicación móvil ejecutándose en emuladores de *tablet* Android (Ilustración 58) e iOS (Ilustración 59). Estas últimas, únicamente en modo horizontal, ya que, dada la naturaleza del producto, se fijó únicamente su uso en posición horizontal de la Tablet. Estas pruebas confirman que previo a la entrega de la documentación, el sistema cumple con el requerimiento de compatibilidad.

A continuación, imágenes que demuestran el cumplimiento del requerimiento:

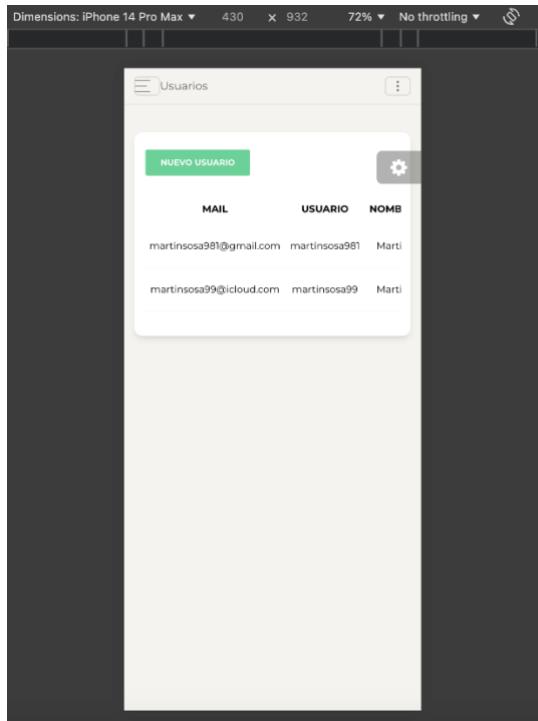


Ilustración 55 - Aplicación web en celular (modo vertical)

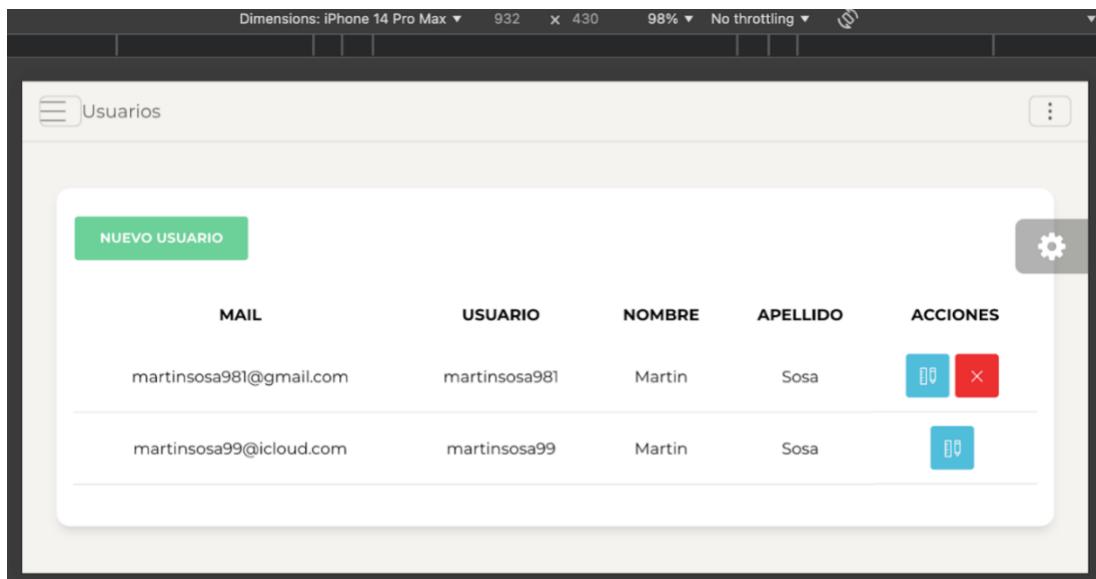


Ilustración 56 - Aplicación web en celular (modo horizontal)

CHEFCHECK

Personal

localhost:4200/#/usuarios

Usuarios

NUEVO USUARIO

MAIL	USUARIO	NOMBRE	APELLIDO	ACCIONES
franciscoecabanillas@gmail.com	fcabanillas	Francisco	Cabanillas	

Ilustración 57 - Aplicación web en computadora

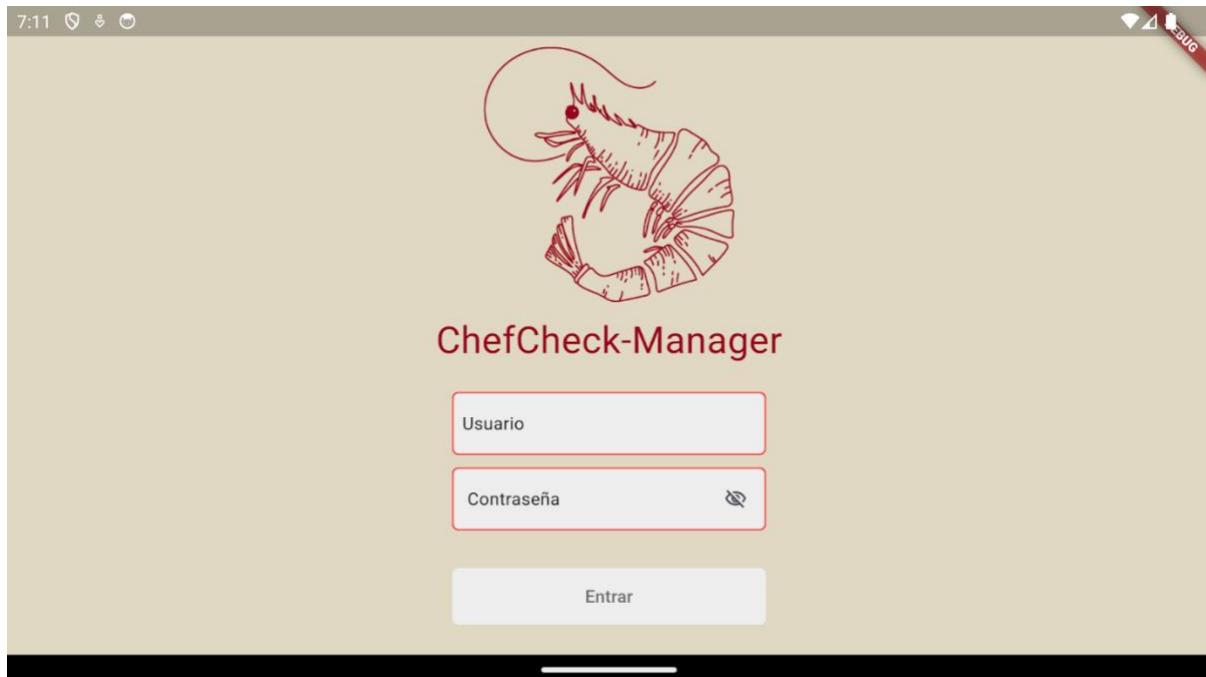


Ilustración 58 - Aplicación móvil en Android

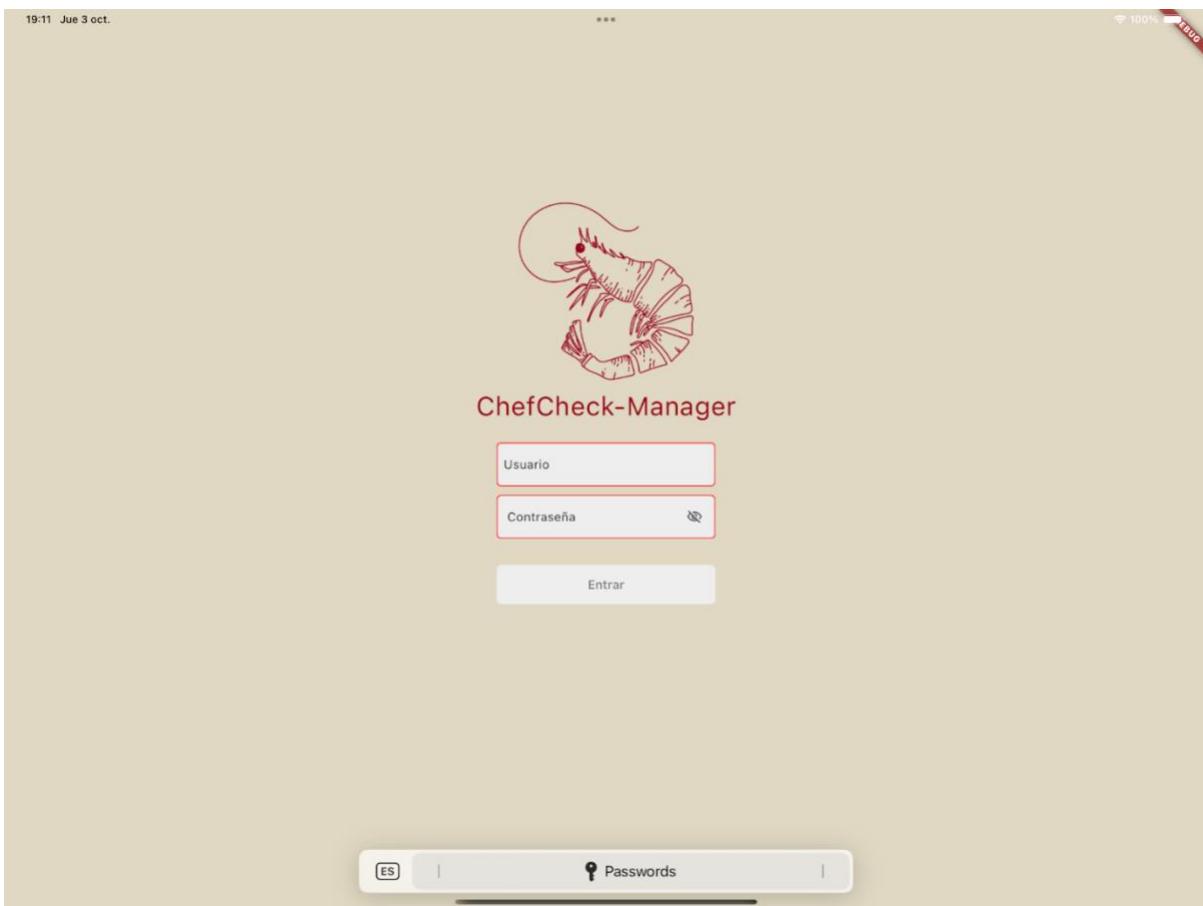


Ilustración 59 - Aplicación móvil en iOS

11.3.2.7. RNF-7 – Extensibilidad

El sistema está diseñado con una arquitectura modular que permite la integración de nuevos módulos y funcionalidades en el futuro. Si bien no se ha implementado una solución de *machine learning* en esta fase, la arquitectura del sistema permitiría su integración de manera eficiente, asegurando que futuros desarrollos puedan incorporarse sin grandes cambios en la estructura actual. Esta extensibilidad se logra gracias a la separación lógica de componentes y la capacidad de agregar módulos nuevos de manera aislada.

Esto se sostiene gracias a: 1) Cada componente físico de la arquitectura, tiene su propia responsabilidad. 2) El desacoplamiento que existe en los módulos de la *API*, los cuales permiten extender sin problemas el sistema. Se mencionó anteriormente el uso de patrones e inyección de dependencias, los cuales favorecen este requerimiento no funcional. 3) Grupo de recursos que existe en la infraestructura de Azure, el cual facilita la agregación de nuevos recursos en caso de ser necesario (lo será para en un futuro agregar todo lo necesario para la parte de *machine learning*, como ser su base de datos dedicada)

11.3.2.8. RNF-8 – Mantenibilidad

La infraestructura modular del sistema facilita su mantenimiento y actualización. El código ha sido documentado de acuerdo con estándares de codificación reconocidos (información previamente mencionada en el documento), asegurando que cualquier desarrollador pueda comprender y modificar el código con facilidad. Además, las actualizaciones del sistema pueden realizarse durante períodos de baja actividad en el hotel, minimizando el impacto en los usuarios. Las actualizaciones pueden realizarse sin realizar grandes mejoras en el servicio, gracias a la infraestructura en Azure.

11.3.2.9. RNF-9 – Recuperación ante fallos

Como el sistema de alojará en Azure, eso le garantiza al equipo y al cliente, altos niveles de recuperación ante fallos. Azure proporciona respaldos automáticos y regulares de los datos, así como opciones para la recuperación en caso de problemas críticos.

11.3.2.10. RNF-10 – Integración

El sistema se ha integrado correctamente con los servicios externos de FacturaLista, permitiendo la descarga automática y el ingreso de datos de facturación electrónica. Esta integración se ha probado y funciona sin inconvenientes, cumpliendo así con los requisitos planteados para esta funcionalidad. Este documento previamente detalló el cumplimiento y detalle de dicha integración.

11.3.2.11. RNF-11 – Monitorización y registro

El sistema incluye un módulo de monitorización que registra los errores ocurridos en una base de datos de logs. Cada vez que ocurre una excepción, se almacenan detalles como el tipo de error, la fecha y el usuario afectado. Esta información es crucial para el equipo de desarrollo, ya que permite un diagnóstico rápido y una solución eficiente de los problemas detectados. Previamente en el documento, también se habló de esto, donde se da a entender, que el requerimiento se pudo cumplir previo a la entrega del proyecto.

11.3.3. Las diez heurísticas de Nielsen

Visibilidad del estado del sistema

Es esencial que el sistema mantenga al usuario informado de manera constante sobre lo que está ocurriendo, proporcionando retroalimentación clara y comprensible en tiempo real. Esto evita que los usuarios se sientan perdidos o desorientados [12].

Compatibilidad entre el sistema y el mundo real

El sistema debe comunicarse usando términos y conceptos que los usuarios ya conozcan del mundo real. Esto ayuda a que interactúen de forma intuitiva y comprendan rápidamente cómo utilizarlo [12]..

Control y libertad del usuario

Los usuarios deben tener la opción de deshacer y rehacer acciones fácilmente. Esta capacidad les proporciona confianza y control, permitiéndoles corregir errores sin temor a consecuencias irreversibles [12]..

Consistencia y estándares

El sistema debe seguir convenciones y patrones ya establecidos para facilitar la experiencia del usuario. Esto asegura que puedan predecir cómo funcionarán los elementos de la interfaz sin necesidad de aprendizaje adicional [12]..

Prevención de errores

Es más efectivo diseñar el sistema de manera que minimice las posibilidades de error, en lugar de simplemente reaccionar cuando estos ocurren. Prevenir errores desde el diseño mejora la usabilidad [12]..

Reconocer antes que recordar

Los elementos clave deben estar visibles y accesibles, de modo que el usuario no tenga que recordar información de una pantalla a otra. Esto reduce la carga cognitiva y mejora la experiencia general [12]..

Flexibilidad y eficiencia de uso

El sistema debe ofrecer diferentes caminos para realizar tareas, adaptándose tanto a usuarios novatos como avanzados. Los usuarios experimentados deberían contar con atajos o configuraciones que aceleren su interacción [12].

Estética y diseño minimalista

El diseño debe ser limpio y claro, evitando elementos innecesarios o irrelevantes que puedan distraer al usuario. Un enfoque minimalista permite que los usuarios se concentren en lo esencial sin sobrecargas visuales [12].

Ayudar a los usuarios a reconocer, diagnosticar y recuperarse de errores

Los mensajes de error deben ser claros y ayudar al usuario a entender qué sucedió y cómo solucionarlo. De esta manera, se facilita la recuperación de errores sin frustración [12]..

Ayuda y documentación

Aunque un buen diseño debería ser lo suficientemente intuitivo, siempre debe haber disponible una documentación accesible y clara para que los usuarios puedan resolver dudas de manera rápida y eficiente [12].

11.3.4. Cumplimiento de las heurísticas de Nielsen

Visibilidad del estado del sistema

A continuación, podemos visualizar un ejemplo (Ilustración 60) de cómo se le muestra al usuario el estado de carga, para que esté informado de que se está realizando una transacción y tiene que esperar:

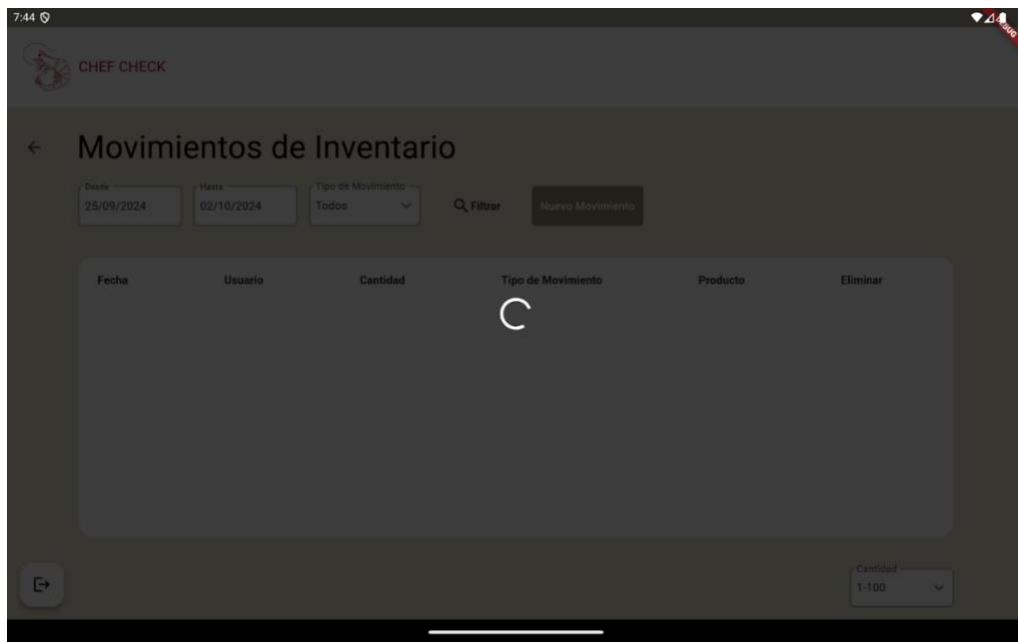


Ilustración 60 - Estado de carga en aplicación móvil

Compatibilidad entre el sistema y el mundo real

A continuación, podemos visualizar cómo se cumple esa heurística, donde el lenguaje utilizado en el sistema coincide con el lenguaje de negocio que el cliente utiliza. Por ejemplo, en la aplicación móvil (Ilustración 61), cuando se registra un comensal en el comedor del hotel, se debe elegir el tipo de comida o turno al cual siesta haciendo referencia (desayuno, almuerzo, merienda o cena). Se podría haber mostrado esas opciones en inglés, tal cual el sistema las almacena en la base de datos, pero para un correcto entendimiento del usuario, lo lógico es mostrar el lenguaje de negocio.

Algo similar ocurre en la (Ilustración 62), donde se aprecia que el lenguaje utilizado es el de negocio, y no se utilizan términos técnicos o ajenos al hotel.

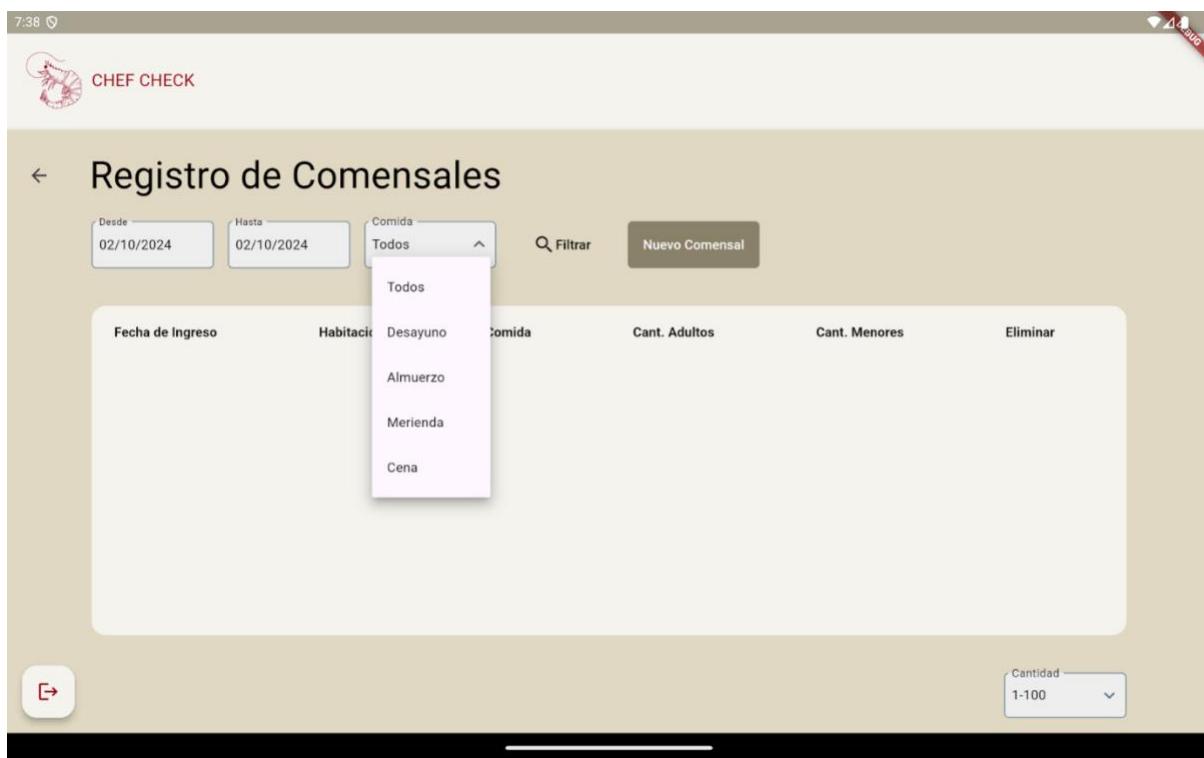


Ilustración 61 - Ejemplo de lenguaje de negocio en aplicación móvil

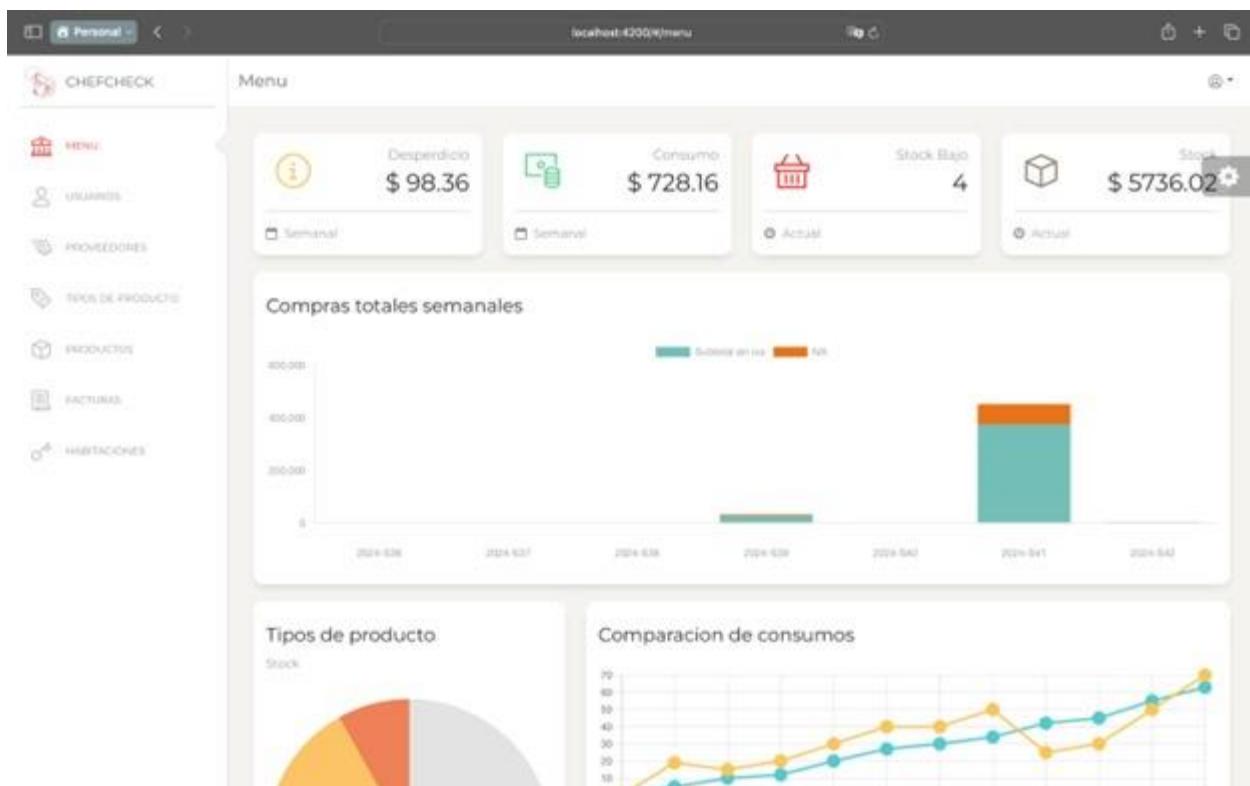


Ilustración 62 – Ejemplo de lenguaje de negocio en aplicación web

Control y libertad del usuario

A continuación, podemos visualizar ejemplos de control y libertad del usuario, donde se observan botones para volver hacia atrás (Ilustración 63), si es que el usuario entró sin querer a dicha pantalla, y un botón de cierre de popup (Ilustración 64), en caso de abrirlo por error.

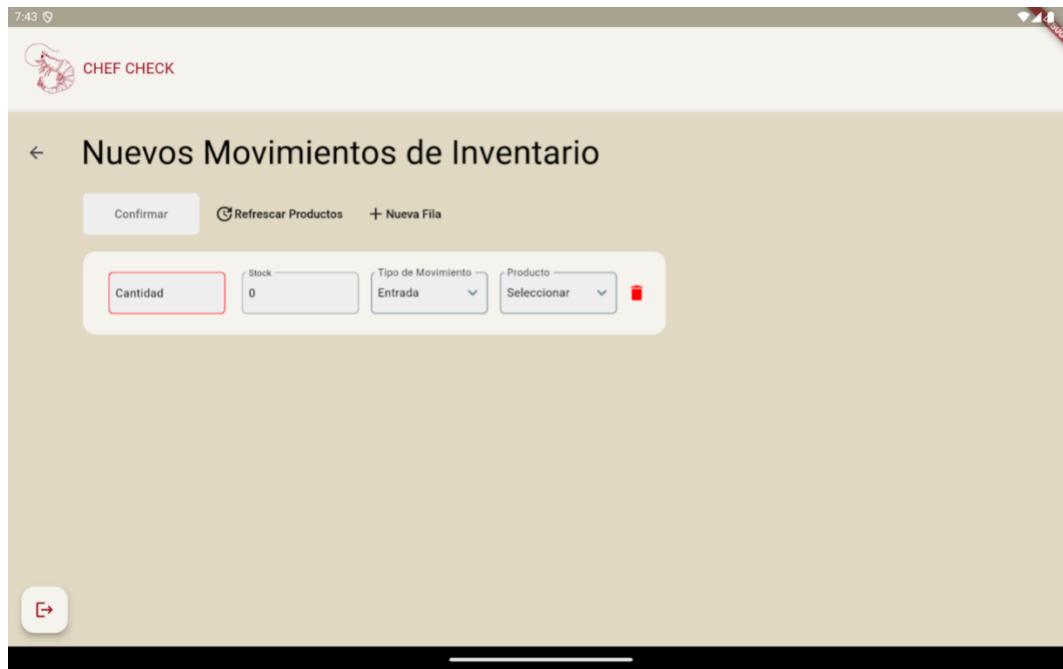


Ilustración 63 - Ejemplo de control y libertad en aplicación móvil

A screenshot of a web-based application interface. On the left, a sidebar lists "PERSONAL", "MENU", "USUARIOS", "PROVEEDORES", "TIPOS DE PRODUCTO", "PRODUCTOS", "FACTURAS" (which is highlighted in red), and "HABITACIONES". The main area shows a "Facturas" section with search filters for "Desde: 11/10/2024" and "Hasta: 20/10/2024", and a "BUSCAR EN FACTURA LISTA" button. A modal window titled "NUEVA FACTURA" is open, showing a receipt from "Finesa S.A." with a list of items. The receipt details include:

RUT EMISOR: 216450470015
TIPO DOCUMENTO: e-Factura
SERIE: A NÚMERO: 697617
FORMA DE PAGO: Credit MONEDA: Peso uruguayo
FECHA DE DOCUMENTO: 11/10/2024

CANT	NOMBRE	PU	DESC	IMPORTE
3	LAMPAZO GOMA EVA	98.36	0	295.08
5	ESCOBA DOBLE BARRIDO	124.59	0	622.95
1	MERMELADA DURAZNO LOS NIETITOS 0% 340G	113.6	0	113.6
24	LECHE DESCREMADA CONAPROLE 1L	60.68	0	1456.32
24	LECHE ENTERA CALCAR 1L	60.68	0	1456.32

At the bottom of the receipt, there are summary lines:

Subtotal gravado mínimo:	2987.72	Total IVA mínimo:	298.77
Subtotal gravado básico:	24866.49	Total IVA básico:	5470.63
Subtotal no gravado:	27854.21	Total a pagar:	33623.61

The right side of the screen shows a "ACCIONES" (Actions) panel with several blue and red buttons.

Ilustración 64 - Ejemplo de control y libertad en aplicación web

Consistencia y estándares

En las siguientes cuatro imágenes (Ilustración 65, Ilustración 66, Ilustración 67 e Ilustración 68), se observa la consistencia y estándares utilizados dentro de cada aplicación. Si bien son dos pantallas diferentes, gracias a su similar diseño y estándares, parecen las mismas.

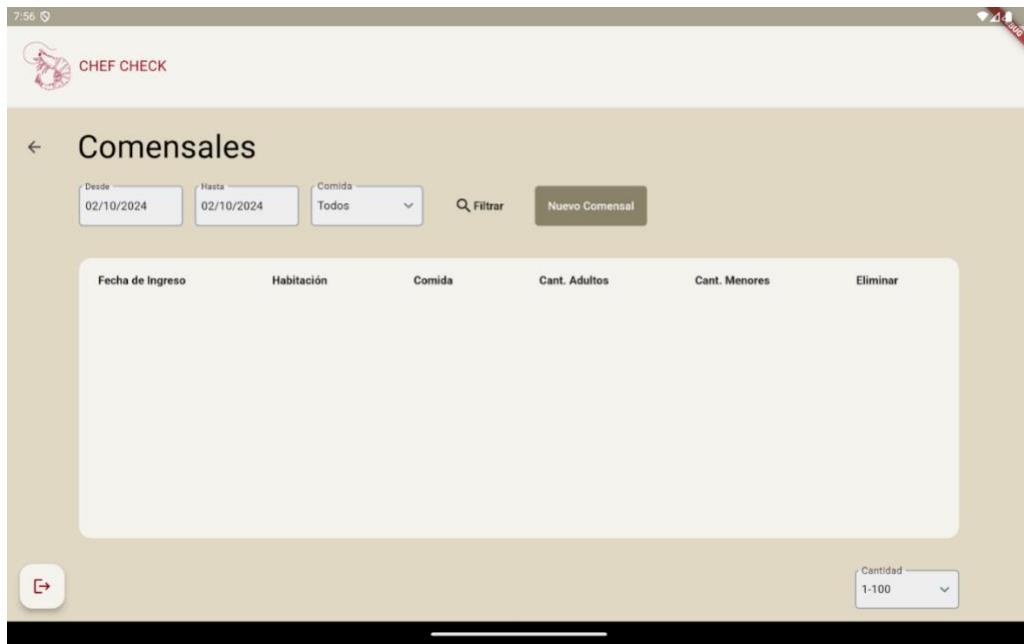


Ilustración 65 - Ejemplo 1 consistencia y estandares (Móvil)

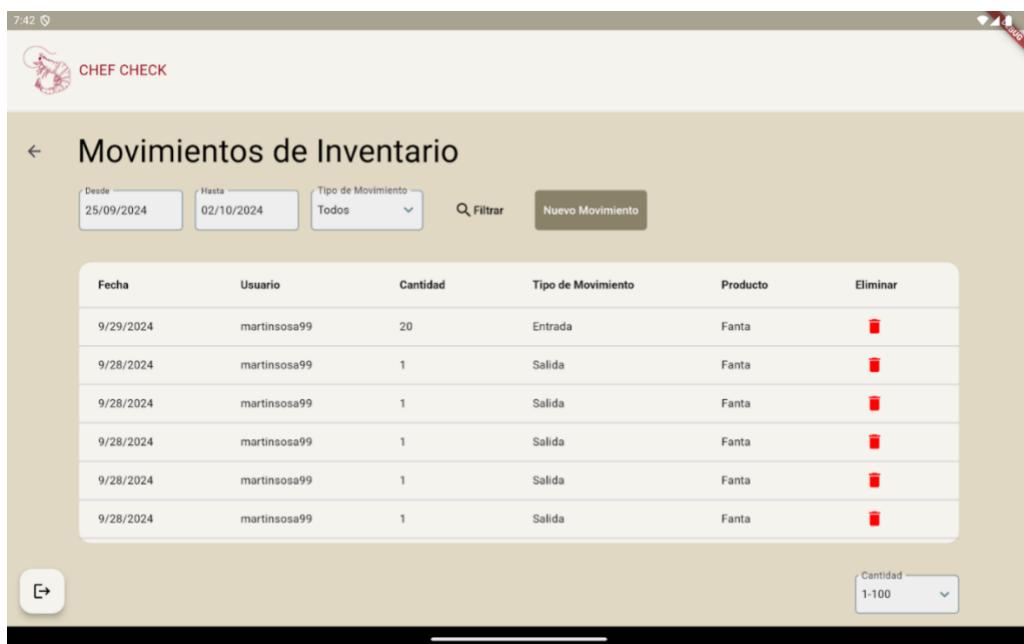


Ilustración 66 - Ejemplo 2 consistencia y estandares (Móvil)

CHEFCHECK

Productos

NUEVO PRODUCTO

NOMBRE	TIPO DE PRODUCTO	PROVEEDOR	STOCK ACTUAL	STOCK MÍNIMO	ACTIVO	ACCIONES
ACEITE ALTO OLEICO TERRA NOSTRA 5L	Almacen	Finesa S.A.	0	5	<input checked="" type="checkbox"/>	
ACEITE DE OLIVA TERRA NOSTRA 3L	Almacen	Finesa S.A.	0	0	<input checked="" type="checkbox"/>	
ACEITE GIRASOL SARANDI 5L	Productos sin asignar	Finesa S.A.	0	1	<input checked="" type="checkbox"/>	
ACEITUNA VERDE DESC 1KG	Productos sin asignar	Finesa S.A.	0	0	<input checked="" type="checkbox"/>	
AJIES VERDES KG	Productos sin asignar	Mercado Miguel	0	0.5	<input checked="" type="checkbox"/>	
AJO EN GRANO 1KG	Productos sin asignar	Finesa S.A.	0	0	<input checked="" type="checkbox"/>	
ALBAHACA BOLSITA	Productos sin asignar	Mercado Miguel	10	3	<input checked="" type="checkbox"/>	
ALCOHOL 70% 950ML	Productos sin asignar	Finesa S.A.	0	0	<input checked="" type="checkbox"/>	
ALCOHOL RECTIFICADO 10LT	Productos sin asignar	Finesa S.A.	0	0	<input checked="" type="checkbox"/>	
	Productos sin asignar	Mercado				

Ilustración 67 - Ejemplo 1 consistencia y estandares (Web)

CHEFCHECK

Tipos De Producto

NUEVO TIPO DE PRODUCTO

NOMBRE	ACTIVO	ACCIONES
Almacen	<input checked="" type="checkbox"/>	
Frutas	<input checked="" type="checkbox"/>	
Lacteos	<input checked="" type="checkbox"/>	
Pescados y Mariscos	<input checked="" type="checkbox"/>	
Productos sin asignar	<input checked="" type="checkbox"/>	
Verduras	<input checked="" type="checkbox"/>	

Ilustración 68 - Ejemplo 2 de consistencia y estandares (Web)

Prevención de errores

Si bien es útil atajar todos los tipos de errores para mostrar una respuesta, es mejor directamente no permitir que pueda haber un error.

A continuación, se observa una imagen de un calendario (Ilustración 69), el cual es utilizado para setear las fechas de los campos "Desde" y "Hasta" (arriba a la izquierda). Las reglas de negocio nos dicen que "Hasta" no puede ser menor a "Desde", y, por ende, si el usuario intenta quebrar esa regla, el selector de fechas no lo deja, impidiendo que exista el error.

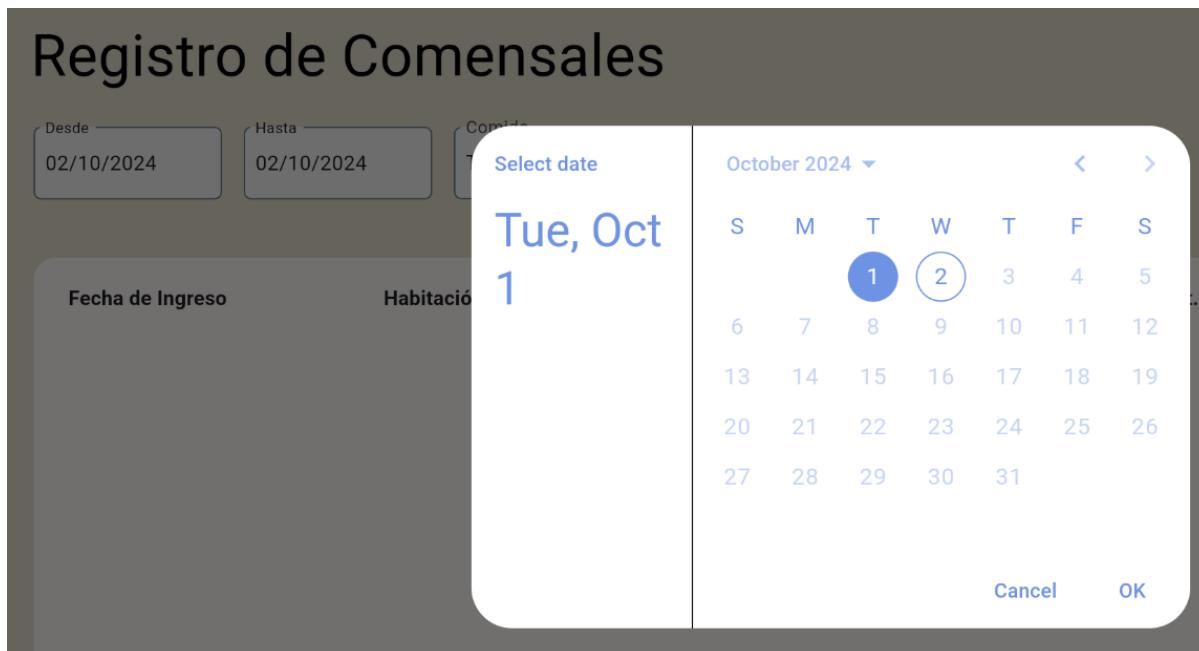


Ilustración 69 - Ejemplo de prevención de errores (Móvil)

A continuación, se observa una imagen del registro de productos del panel *web* (Ilustración 70). Para prevenir errores en el tipo de producto, se dispone de un dropdown con únicamente los valores válidos. De esa manera, al no colocar un campo de texto libre, se impide el error del usuario directamente.

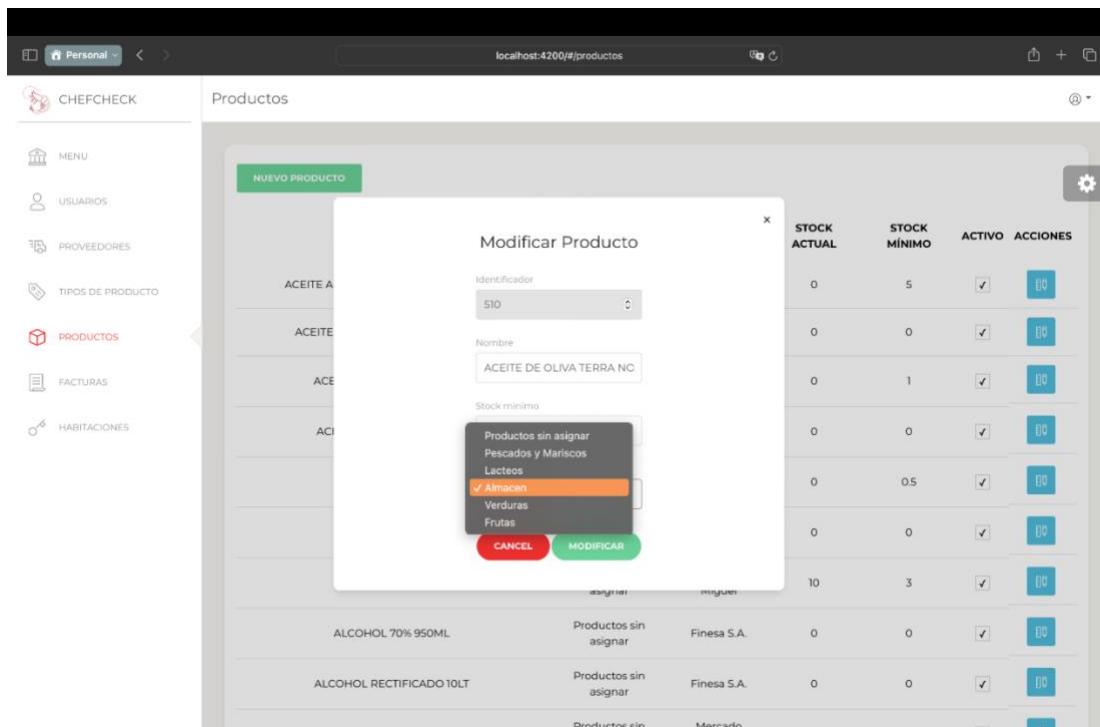


Ilustración 70 - Ejemplo de prevención de errores (Web)

Reconocer antes que recordar

Se buscó que el usuario no tenga que llenar ningún dato donde sus valores sean previamente conocidos o limitados, como es el caso de los dropdowns utilizados (Ilustración 71 e Ilustración 72).

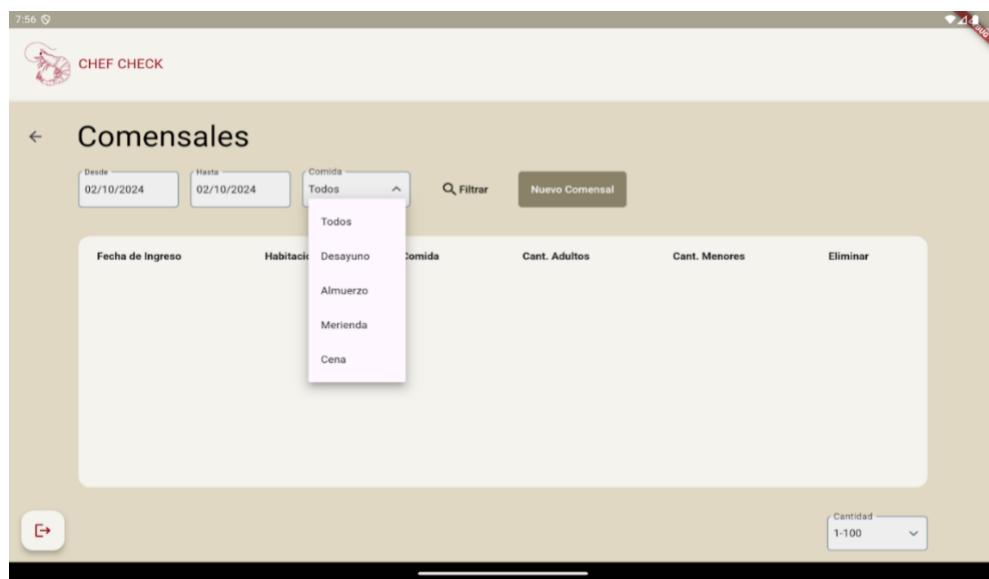


Ilustración 71 - Ejemplo de reconocer antes que acordar (Móvil)

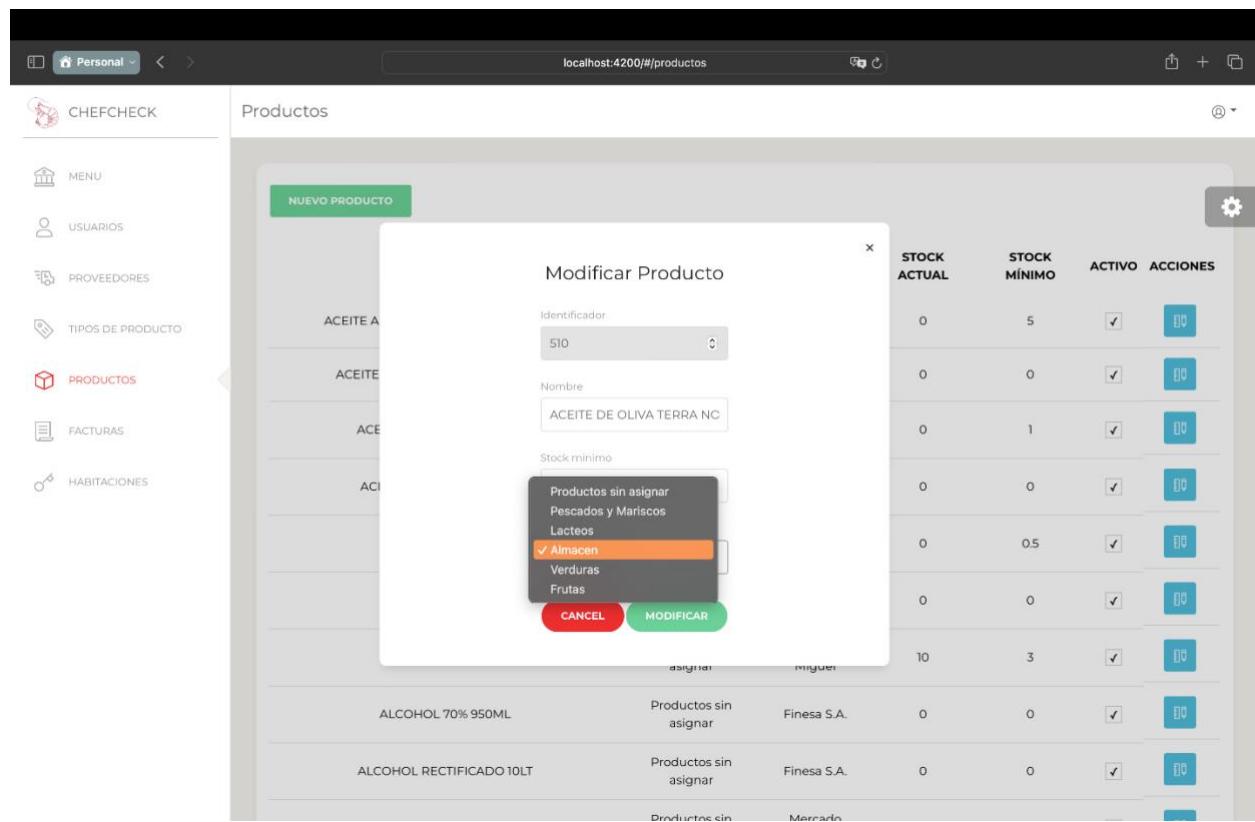


Ilustración 72 - Ejemplo de reconocer antes que recordar (Web)

Flexibilidad y eficiencia de uso

En ambas aplicaciones (*web* y móvil), no se cumple esta heurística, ya que, aunque hay diferentes roles de usuario, todos interactúan con la interfaz de manera similar, sin distinción entre usuarios básicos y avanzados. No se implementó la posibilidad de personalizar la experiencia para optimizar tareas, ya que no hay un concepto de usuario pro o experto que requiera estas funciones. Todos los usuarios siguen los mismos flujos y opciones, sin posibilidad para ajustar la aplicación según sus preferencias.

Estética y diseño minimalista

Se buscó que el diseño no esté sobrecargado de elementos innecesarios, donde la atención del usuario se enfoque únicamente en lo relevante.

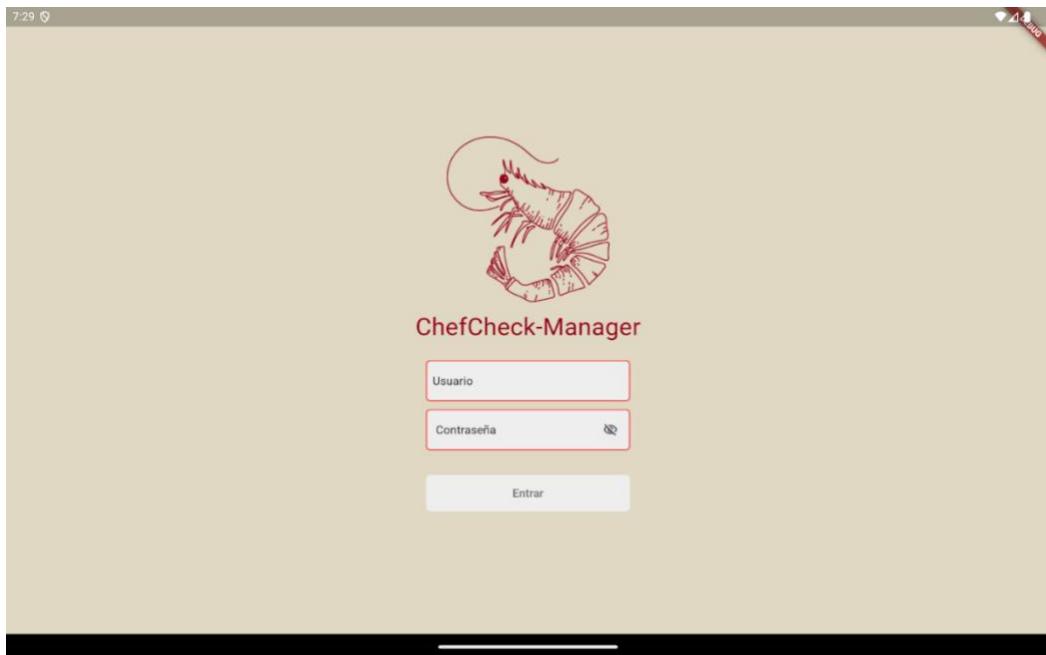


Ilustración 73 - Ejemplo de estética y diseño minimalista (Móvil)

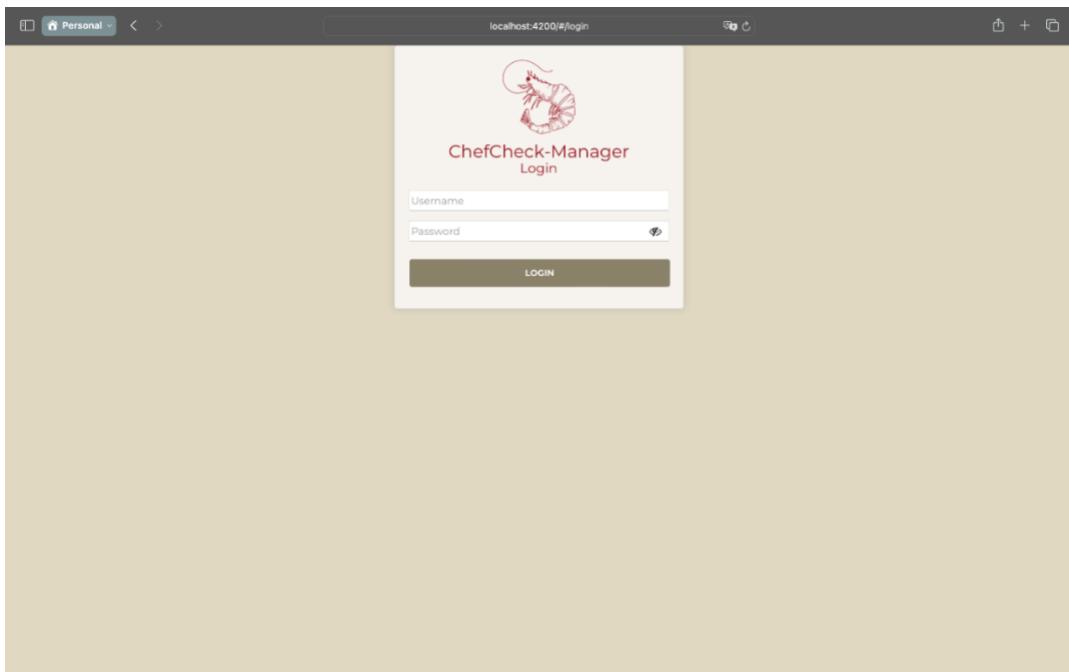


Ilustración 74 - Ejemplo de estética y diseño minimalista (Web)

Ayudar a los usuarios a reconocer, diagnosticar y recuperarse de errores

El equipo dejó mensajes de error claros, para que los usuarios (los cuales no son técnicos), puedan entender la causa o el motivo del error (Ilustración 75 e Ilustración 76).

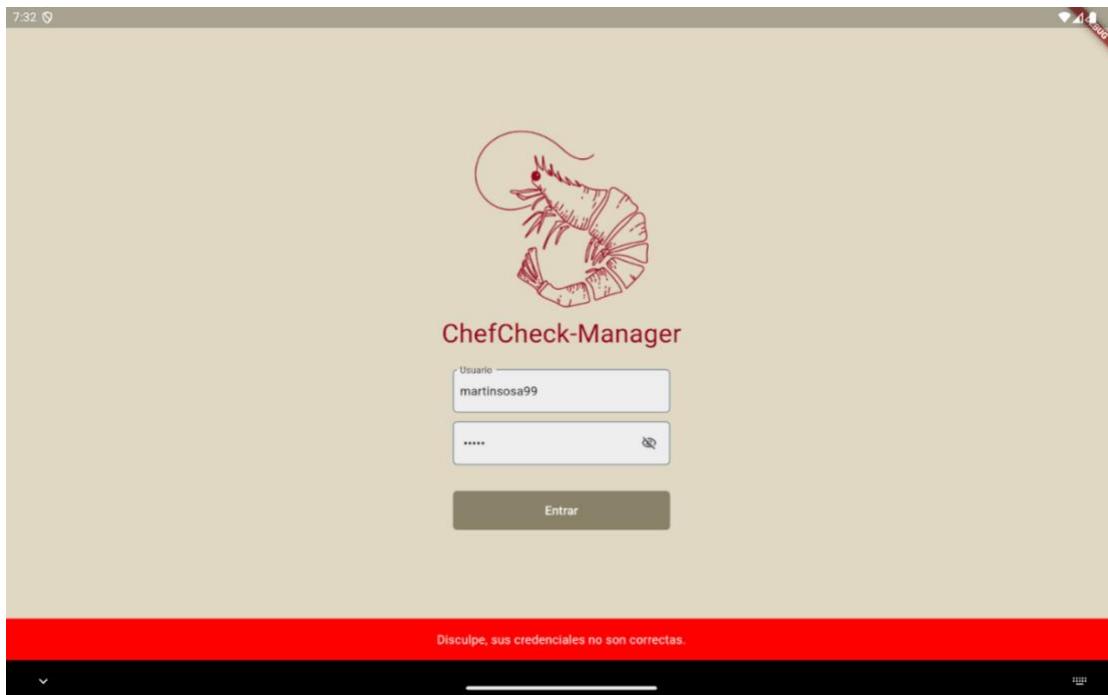


Ilustración 75 - Ejemplo mensaje de error (Móvil)

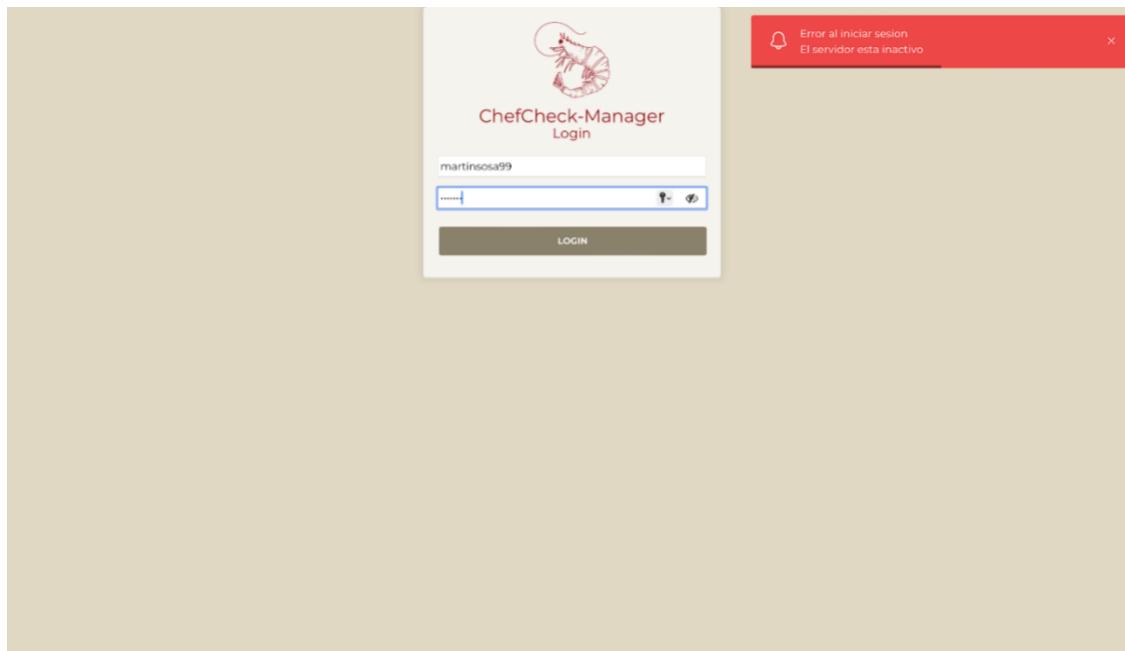


Ilustración 76 - Ejemplo mensaje de error (Web)

Ayuda y documentación

No se aplicó documentación ni secciones de preguntas frecuentes. La idea del equipo fue realizar interfaces de usuario lo suficientemente claras como para prescindir de guías.

11.3.5. Diagramas de clase (por paquete)



Ilustración 77 - Diagrama del paquete CC_Manager_services.BusinessLogic

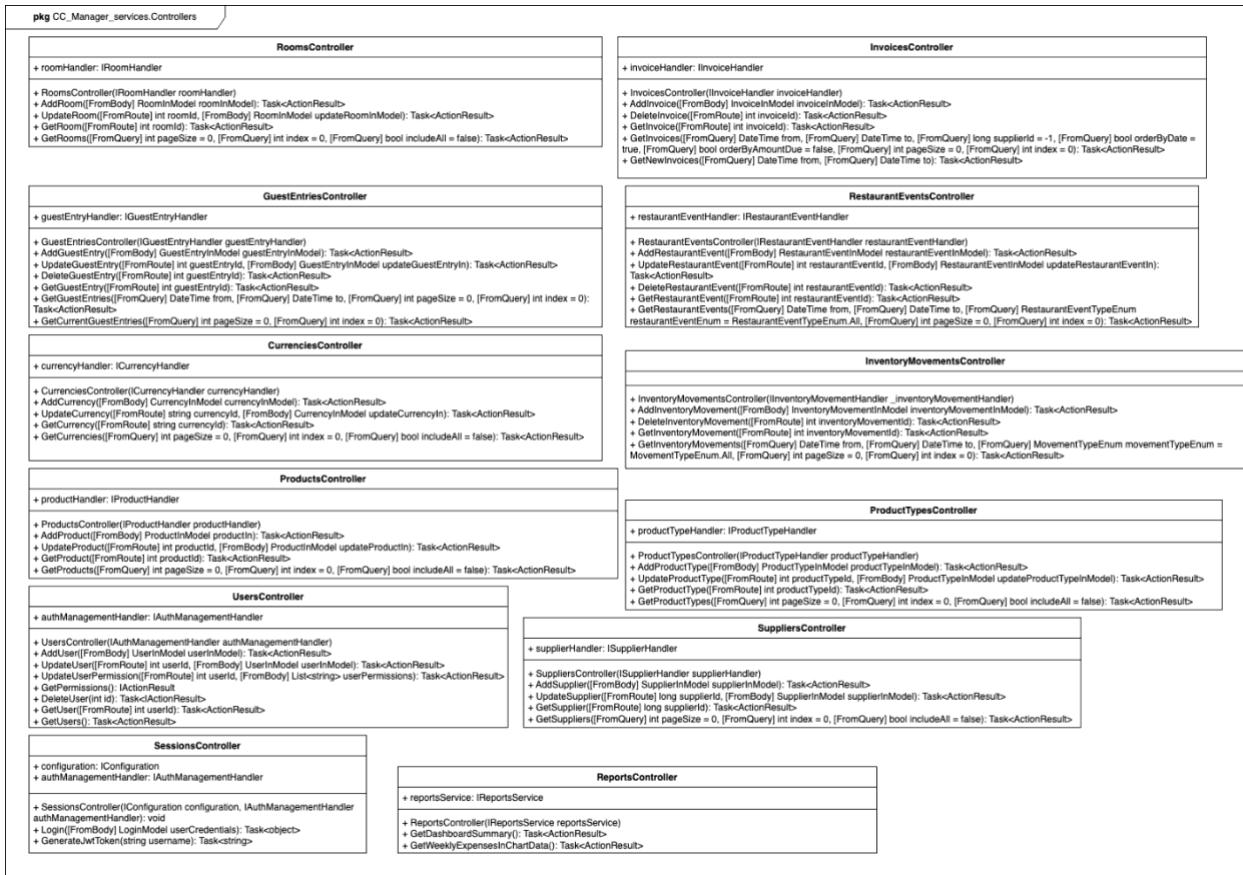


Ilustración 78 - Diagrama del paquete CC_Manager_services.Controllers

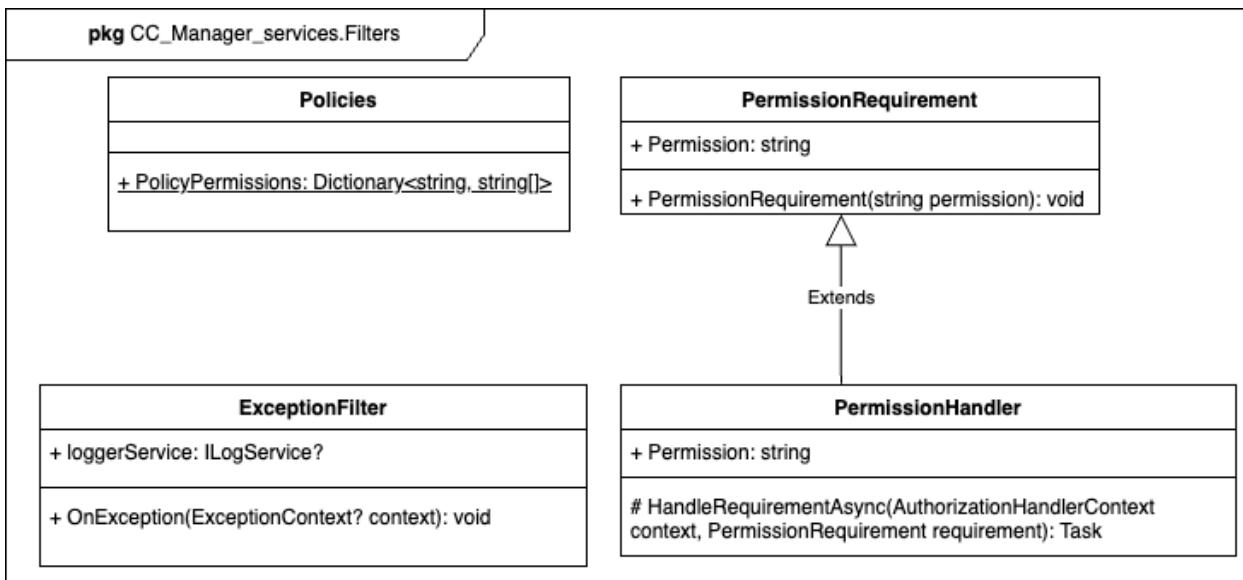


Ilustración 79 - Diagrama del paquete CC Manager services.Filters

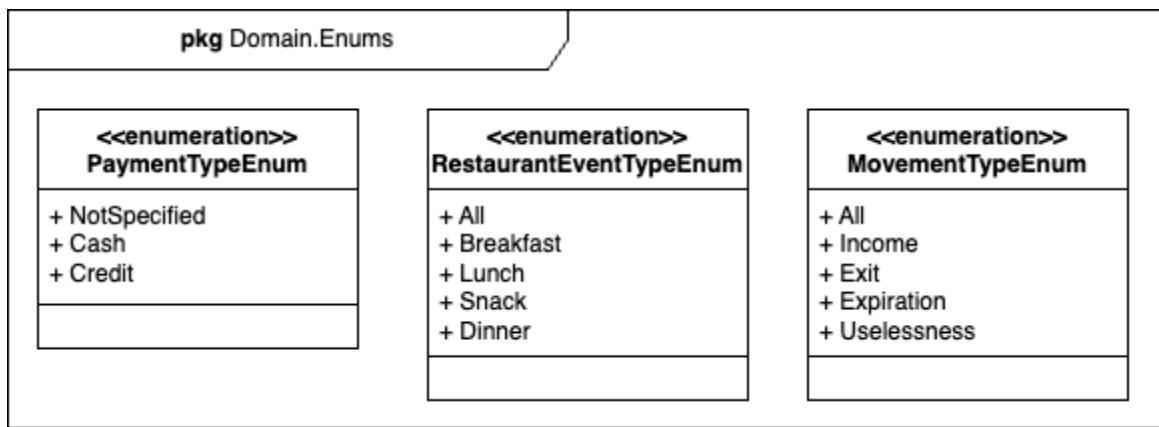


Ilustración 80 - Diagrama del paquete Domain.Enums

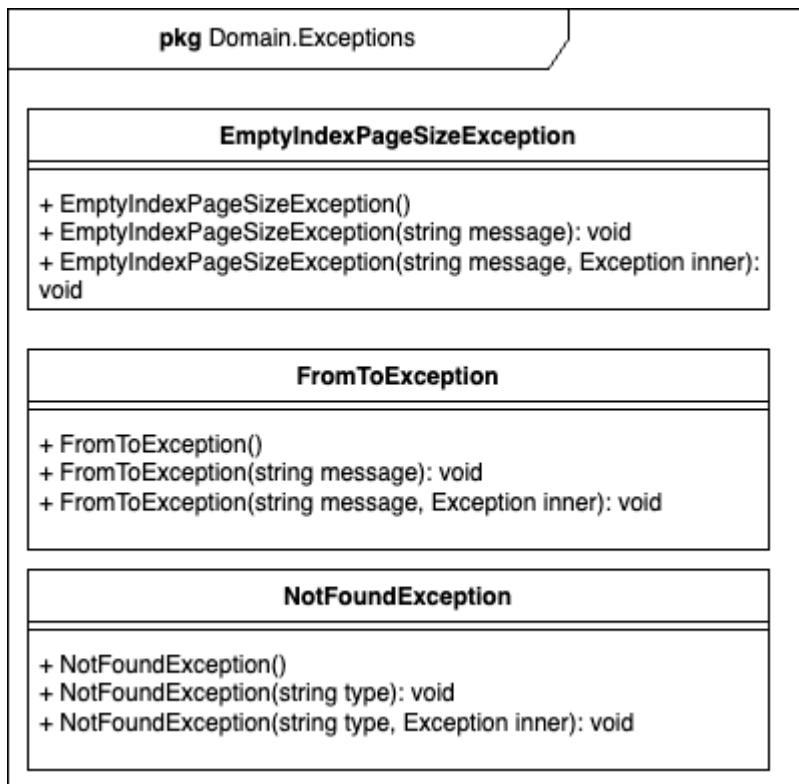


Ilustración 81 - Diagrama del paquete Domain.Exceptions

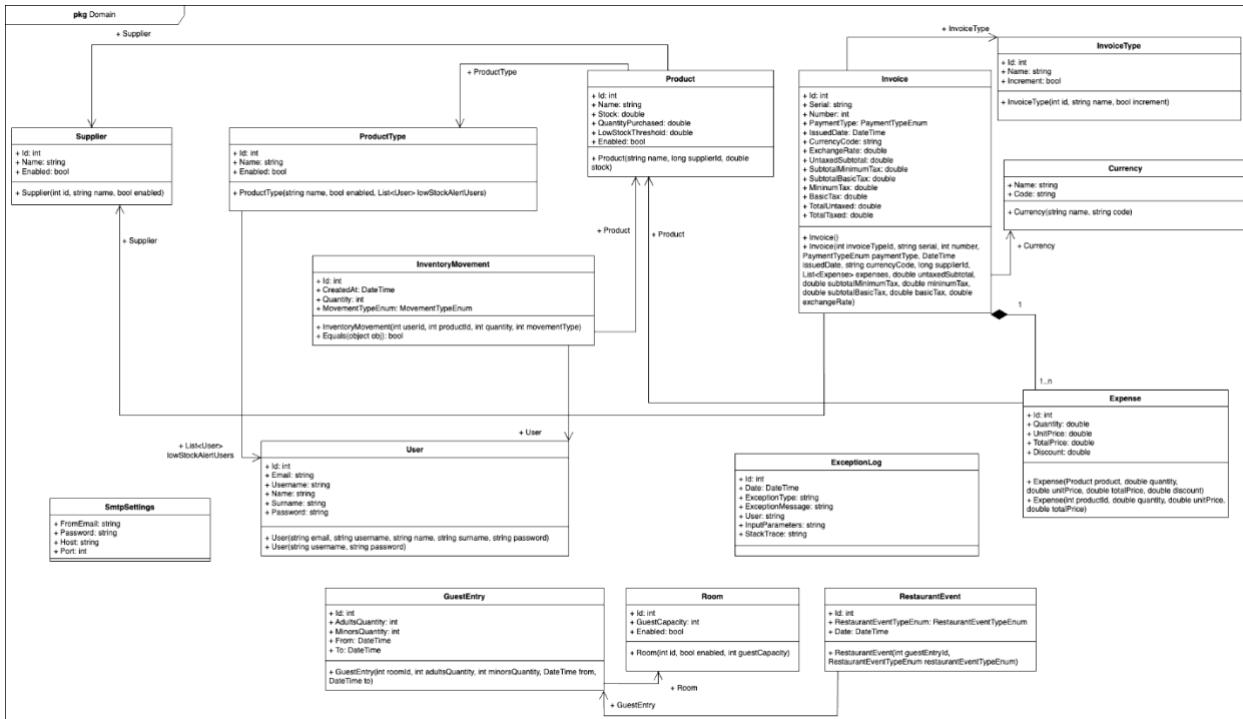


Ilustración 82 - Diagrama del paquete Domain

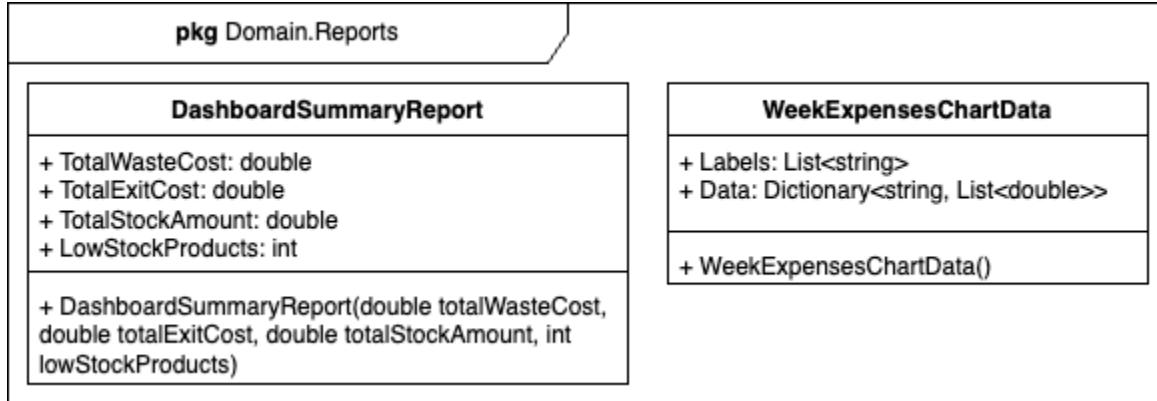


Ilustración 83 - Diagrama del paquete Domain.Reports

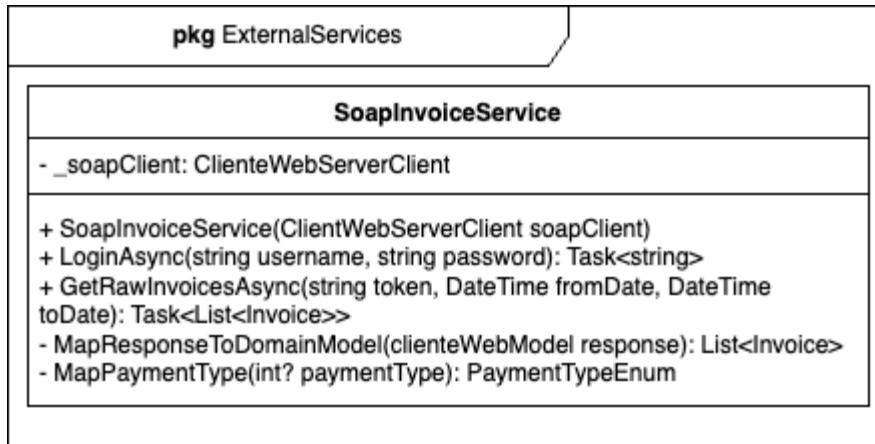


Ilustración 84 - Diagrama del paquete ExternalServices

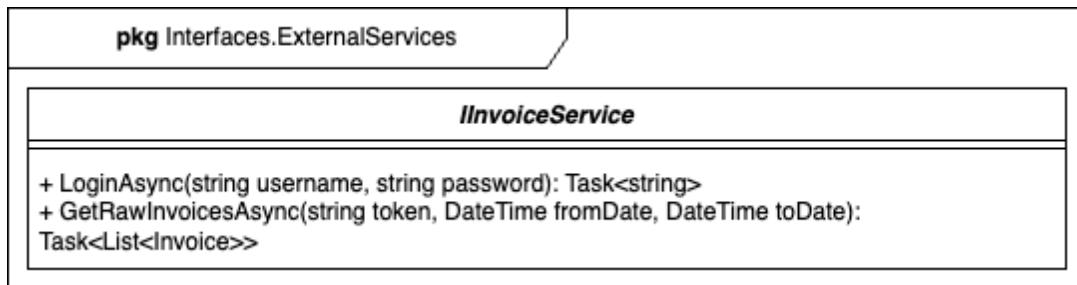


Ilustración 85 - Diagrama del paquete Interfaces.ExternalServices

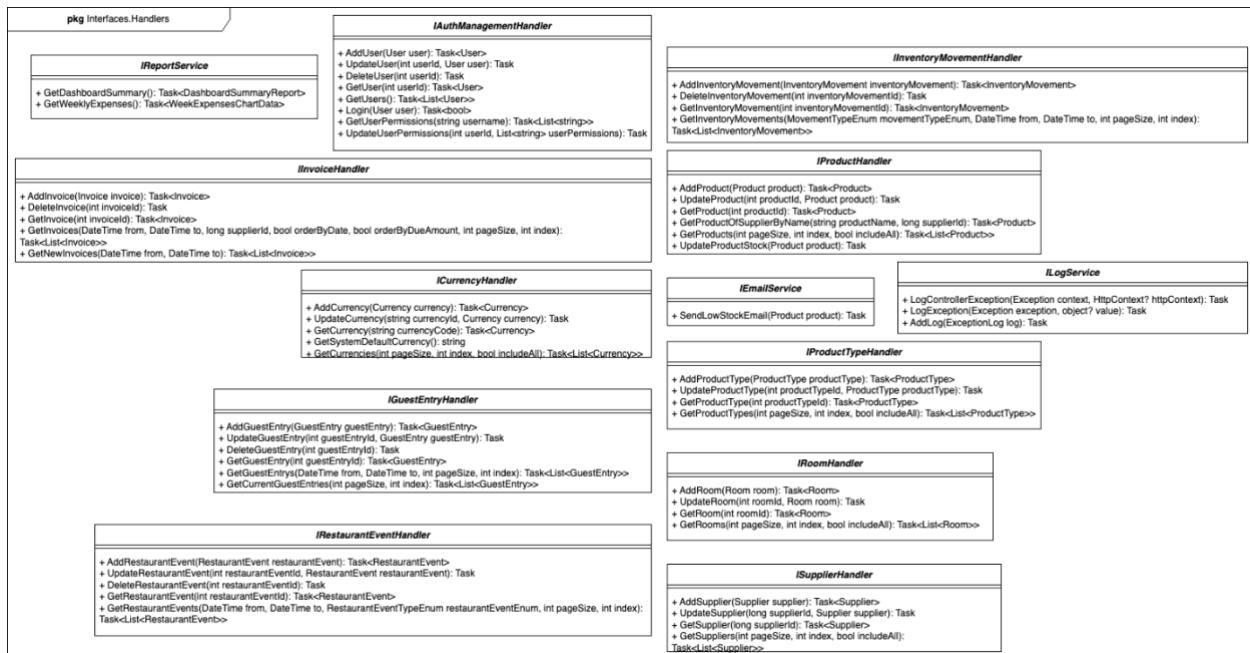


Ilustración 86 - Diagrama del paquete Interfaces.Handlers

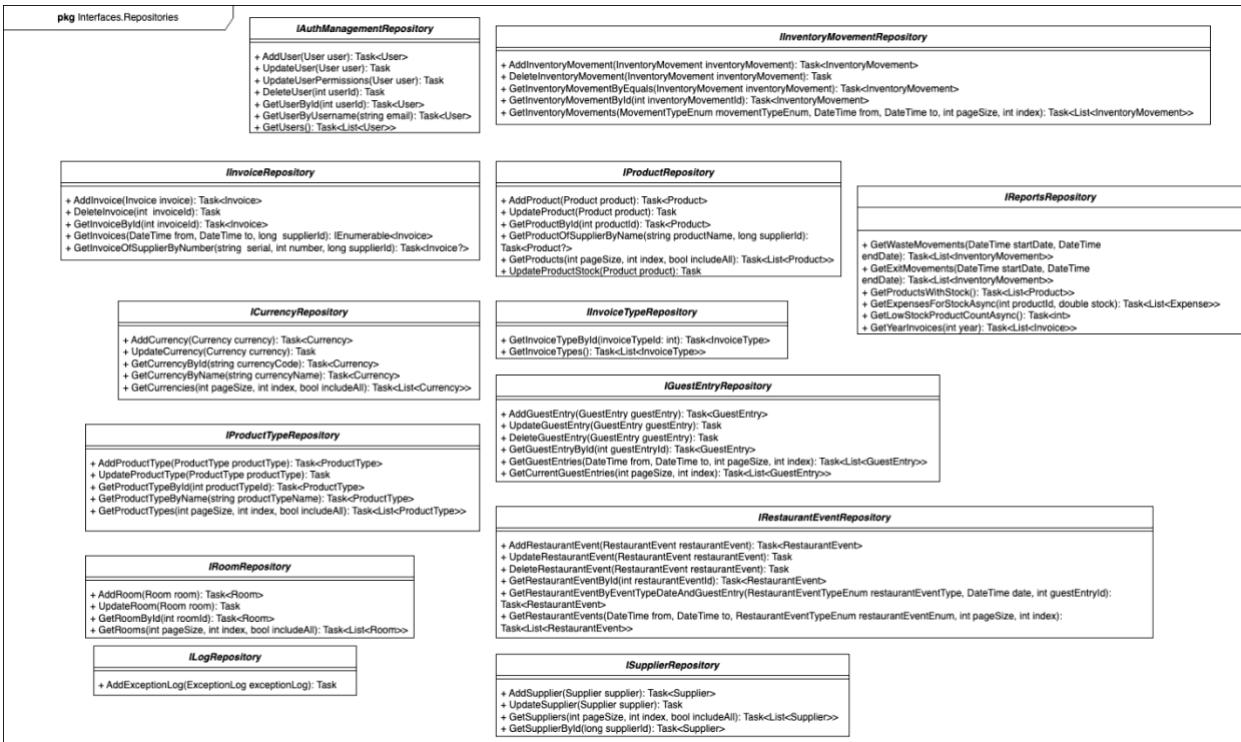


Ilustración 87 - Diagrama del paquete Interfaces.Repositories

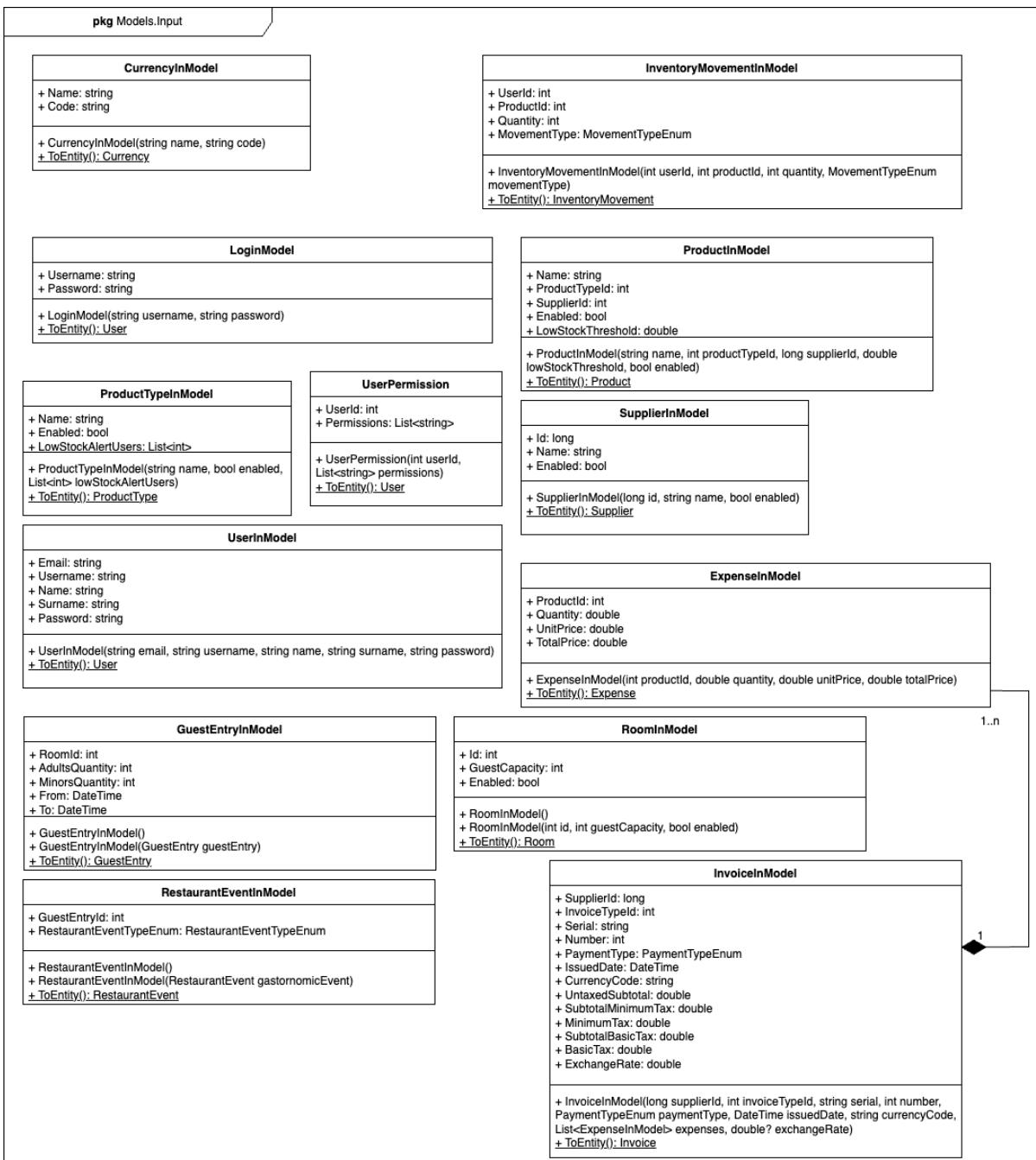


Ilustración 88 - Diagrama del paquete Models.Input

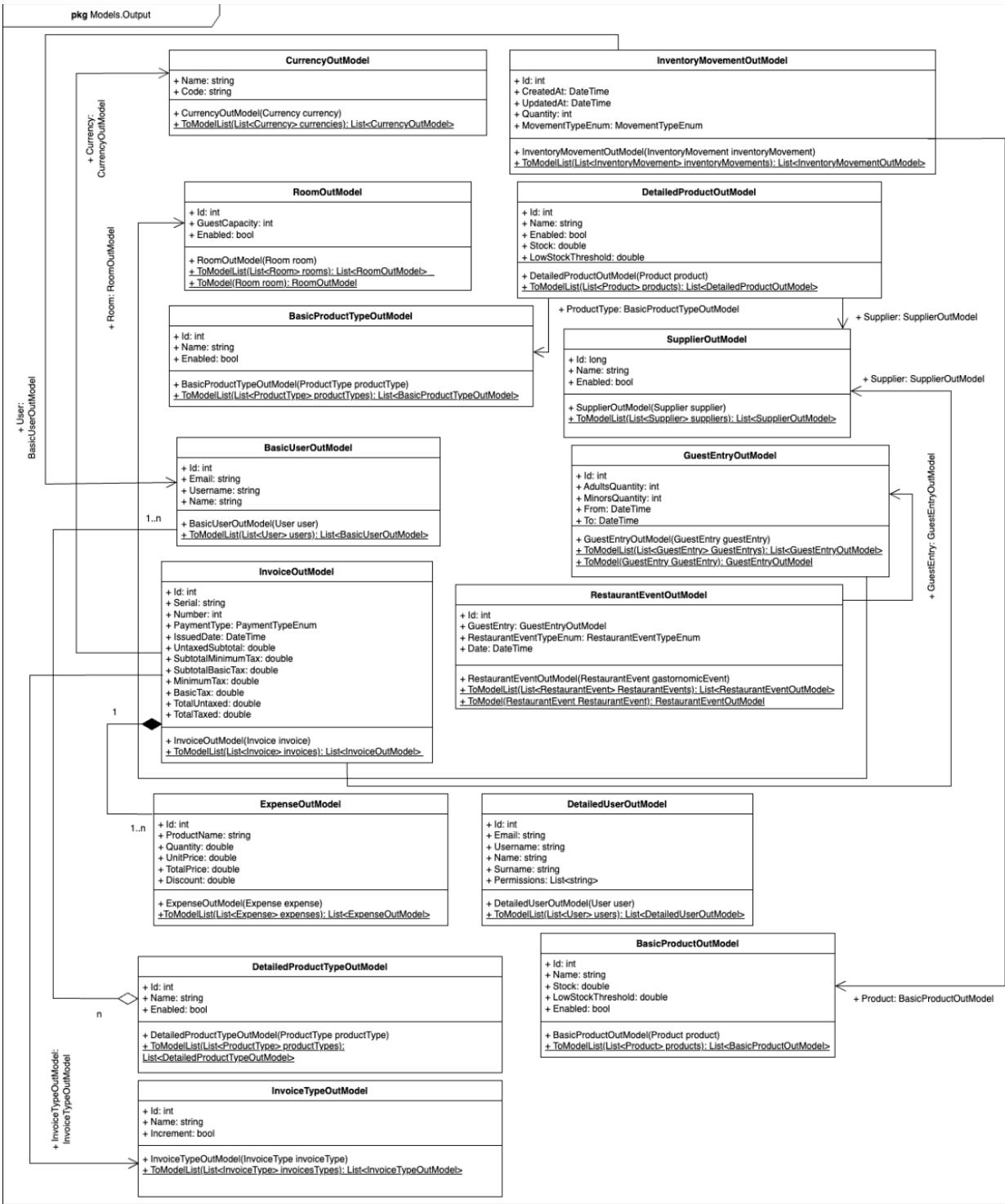


Ilustración 89 - Diagrama del paquete Models.Output

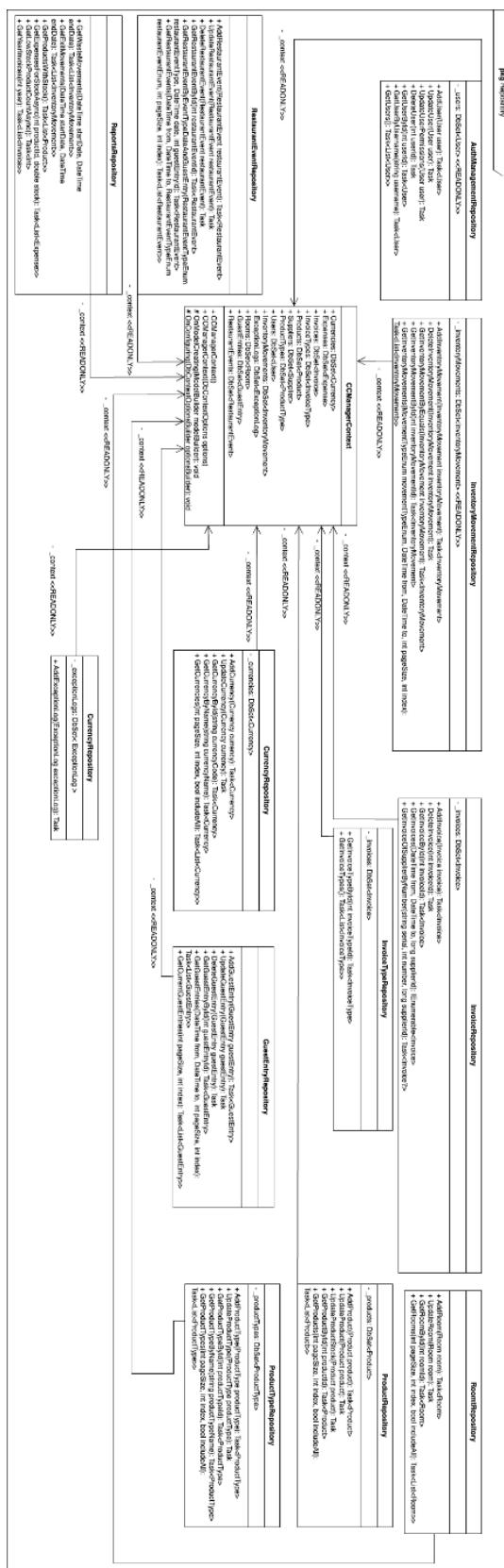


Ilustración 90 - Diagrama del paquete Repository

11.4. Gestión del proyecto

11.4.1. Behaviour Driven Development

Behavior Driven Development (BDD) es una metodología de desarrollo de *software* que se centra en definir el comportamiento deseado de una aplicación a través de ejemplos claros y comprensibles, escritos en un lenguaje común para todas las partes involucradas. A diferencia de enfoques más técnicos como el *test driven development*, BDD se basa en la colaboración entre equipos técnicos y no técnicos, utilizando descripciones que todos pueden entender. Esto facilita la comunicación y asegura que las expectativas del sistema sean claras desde el principio.

Una característica clave de BDD es que las pruebas se escriben antes del desarrollo, pero con un enfoque en cómo debería comportarse el sistema desde la perspectiva del usuario. Esto se logra mediante la redacción de historias de usuario, donde cada escenario se estructura en términos de contexto (*Given*), acción (*When*) y resultado esperado (*Then*). Esta fórmula narrativa ayuda a detallar de manera comprensible las interacciones del usuario con el sistema, lo que asegura que el equipo de desarrollo construya exactamente lo que se necesita, alineando las expectativas de todas las partes.

11.4.2. Gestión del riesgo

11.4.2.1. Probabilidad de ocurrencia

Probabilidad	Descripción
1	Mínima posibilidad de ocurrencia
2	Probabilidad muy baja
3	Probabilidad baja
4	Probabilidad media baja
5	Probabilidad media
6	Probabilidad media alta
7	Probabilidad alta
8	Probabilidad muy alta
9	Se está gestando el problema
10	Es un problema activo

11.4.2.2. Impacto

Valor	Descripción
1	Muy bajo
2	Bajo
3	Medio
4	Alto
5	Muy alto

11.4.2.3. Riesgos

Identificador de riesgo	Riesgo	Tipo de riesgo	Descripción
PRJ-01	Problemas de estimación	1. Proyecto	Los problemas de estimación ocurren cuando el tiempo, los recursos o los costos del proyecto se calculan incorrectamente. Esto puede llevar a retrasos, sobrecostes y una asignación inadecuada de recursos, afectando la viabilidad y calidad del proyecto.
PRJ-02	Tiempos de dedicación personal al proyecto	1. Proyecto	No lograr cumplir con los tiempos mínimos necesarios de dedicación personal al proyecto.
PRJ-03	Obtención de credenciales digitales	1. Proyecto	Demoras o dificultades para obtener las credenciales digitales para las consultas a la API de la DGI.
PRJ-04	Mala distribución de la carga de trabajo	1. Proyecto	La mala distribución de la carga de trabajo puede causar sobrecarga y estrés en algunos miembros del equipo, mientras otros se sienten infravalorados y desmotivados. Esto puede afectar la productividad y comprometer la entrega del proyecto.
PRJ-05	Sobrecostes de implementación	1. Proyecto	El riesgo de sobrecostes de implementación se presenta cuando los gastos del proyecto exceden el presupuesto planificado. Esto puede ocurrir por estimaciones inadecuadas, cambios en el alcance o problemas imprevistos. Los sobrecostes pueden comprometer la viabilidad financiera, generar tensiones con el cliente y afectar la calidad del sistema.
PRJ-06	Resistencia al cambio	1. Proyecto	Que los usuarios finales se resistan o ignoren el uso del nuevo sistema.
PRD-01	Subestimación del alcance del proyecto	2. Producto	La subestimación del alcance del proyecto ocurre cuando se subvaloran las tareas y recursos necesarios para completarlo. Esto puede llevar a retrasos, sobrecostes y una sobrecarga de trabajo para el equipo, comprometiendo la calidad del resultado final.
PRD-02	Cambios en los requerimientos del cliente	2. Producto	Modificaciones o agregados en los requerimientos del cliente para el producto final.
PRD-03	Defectos en el software	2. Producto	Los defectos en el software pueden manifestarse como errores, fallos o malfuncionamientos que afectan su

			rendimiento y funcionalidad. Estos problemas pueden comprometer la calidad del sistema, generar costos adicionales para su corrección y retrasar la entrega del proyecto.
PRD-04	Mala usabilidad y ergonomía	2. Producto	Los problemas de usabilidad y ergonomía ocurren cuando el sistema no es intuitivo ni fácil de usar, lo que puede causar errores, frustración y rechazo por parte de los usuarios. Esto afecta la eficiencia operativa y la satisfacción del cliente.
PRD-05	Entrega incompleta	2. Producto	Una entrega incompleta o insuficiente del producto dejando por fuera los requerimientos mas importantes indicados por el cliente.
EQP-01	Comunicación con la tutora	3. Equipo	Una comunicación ineficiente o inadecuada con la tutora del proyecto puede generar malentendidos, falta de alineación en objetivos y expectativas, y retrasos en la toma de decisiones. Esto puede afectar la calidad del trabajo y el cumplimiento de los plazos.
EQP-02	Comunicación con el cliente	3. Equipo	Que se presente una comunicación interrumpida o ineficiente con el cliente / referente del hotel. Afecta al plan de calidad
EQP-03	Comunicación ineficiente o inadecuada entre miembros	3. Equipo	La comunicación ineficiente o inadecuada entre los miembros del equipo puede llevar a malentendidos, retrasos y errores en el proyecto. Esto puede resultar en una falta de coordinación y cohesión, afectando la productividad y la calidad del trabajo.
EQP-04	Problemas personales de fuerza mayor	3. Equipo	Siempre esta la posibilidad que se presenten imprevistos personales de fuerza mayor en uno o varios integrantes del equipo, siendo su impacto mayor en un equipo de 3 personas.
TEC-01	Desconocimiento o dificultades con las tecnologías	4. Técnico	Existe el riesgo de encontrarnos con dificultades para implementar algun detalle con las tecnologías por no ser especialistas de las mismas
TEC-02	Dificultades con la implementación con DGI u intermediarios	4. Técnico	Es posible que la información brindada por los documentos oficiales de la DGI no sean claros respecto al uso de sus endpoints públicos.
EXT-01	Actualizaciones de normativas legales	5. Externo	Las actualizaciones de normativas legales pueden afectar el desarrollo y la implementación del proyecto. Cambios en las leyes o regulaciones pueden requerir modificaciones en el sistema, aumentando los costos y los plazos, y potencialmente afectando la viabilidad y cumplimiento del proyecto. Es crucial monitorear y adaptarse a las nuevas normativas para asegurar que el sistema cumpla con todos los requisitos legales vigentes.

11.4.2.4. Evidencia del registro de riesgos

Sprint	Identificador de riesgo	Título	Tipo	Descripción	Plan de mitigación	Probabilidad (1-10)	Impacto (1-5)	Nivel de riesgo	Probabilidad (1-10)	Impacto (1-5)	Nivel de riesgo	Probabilidad (1-10)	Impacto (1-5)
					Revisões frequentes do cronograma, repassando entre sprints as estimativas e suas diferenças para o projeto. Executar ações corretivas baseadas nelas para os próximos sprints, aprendendo assim sobre nossa velocidade e dificuldades.	7	4	28	8	4	32	8	
PRJ-01	Problemas de estimación	1. Proyecto		Los problemas de estimación ocurren cuando el tiempo, los recursos o los costos del proyecto se calculan incorrectamente. Esto puede llevar a retrasos, sobrecostos y una asignación inadecuada de recursos, afectando la viabilidad y calidad del proyecto.									
PRJ-02	Tiempos de dedicación personal al proyecto	1. Proyecto		No lograr cumplir con los tiempos mínimos necesarios de dedicación personal al proyecto.	Comunicar y armonizar las dificultades para coordinar cobertura de los requerimientos por parte de otros miembros	8	4	32	8	4	32	9	
PRJ-03	Obtención de credenciales digitales	1. Proyecto		Demoras o dificultades para obtener las credenciales digitales para las consultas a la base de datos.	Una investigación temprana de su necesidad y como obtenerla en caso de ser necesaria	6	5	30	7	5	35	8	
PRJ-04	Mala distribución de la carga de trabajo	1. Proyecto		La mala distribución de la carga de trabajo puede causar sobrecarga y estrés en algunos miembros del equipo, mientras otros se sienten infravalorados y desanimados. Esto puede afectar la productividad y comprometer la entrega del proyecto.	Equilibrar la carga de trabajo dependiendo por el cronograma personal de cada uno, así como también las fortalezas y debilidades de cada miembro	8	5	40	7	4	28	8	
PRJ-05	Sobrecostes de implementación	1. Proyecto		El riesgo de sobrecostes de implementación se presenta cuando los gastos del proyecto exceden los presupuestos planeados. Esto puede ocurrir por estimaciones inadecuadas, cambios en el alcance o problemas imprevistos. Los sobrecostes pueden comprometer la viabilidad financiera, generar tensiones con el cliente y afectar la calidad del sistema.	Evitar el uso de librerías grandes que pueden consumir mucho rendimiento de memoria en la nube o una tablet de gran rendimiento.	7	4	28	6	4	24	6	
PRJ-06	Resistencia al cambio	1. Proyecto		Que los usuarios finales se resistan o ignoren el uso del nuevo sistema.	Gestión del cambio y capacitación	3	3	9	5	3	15	5	
PRD-01	Subestimación del alcance del proyecto	2. Producto		La subestimación del alcance del proyecto ocurre cuando se subestiman las tareas y recursos necesarios para completarlo. Esto puede llevar a retrasos, sobrecostos y una sobrecarga de trabajo para el equipo, comprometiendo la calidad del resultado final.	Realizar una buena planificación de experimentos y en qué Sprint se realizarán cada uno.	5	5	25	5	5	25	6	
				Modificaciones o agregados en los requerimientos del cliente cara el	Analizar la posibilidad y dificultad de los cambios. En casos que requiera mayores esfuerzos comunicarlos al cliente para encontrar un punto								

Ilustración 91 - Evidencia del registro de riesgos en Excel

11.4.3. Gestión de la comunicación

Evidencia de reuniones con el cliente

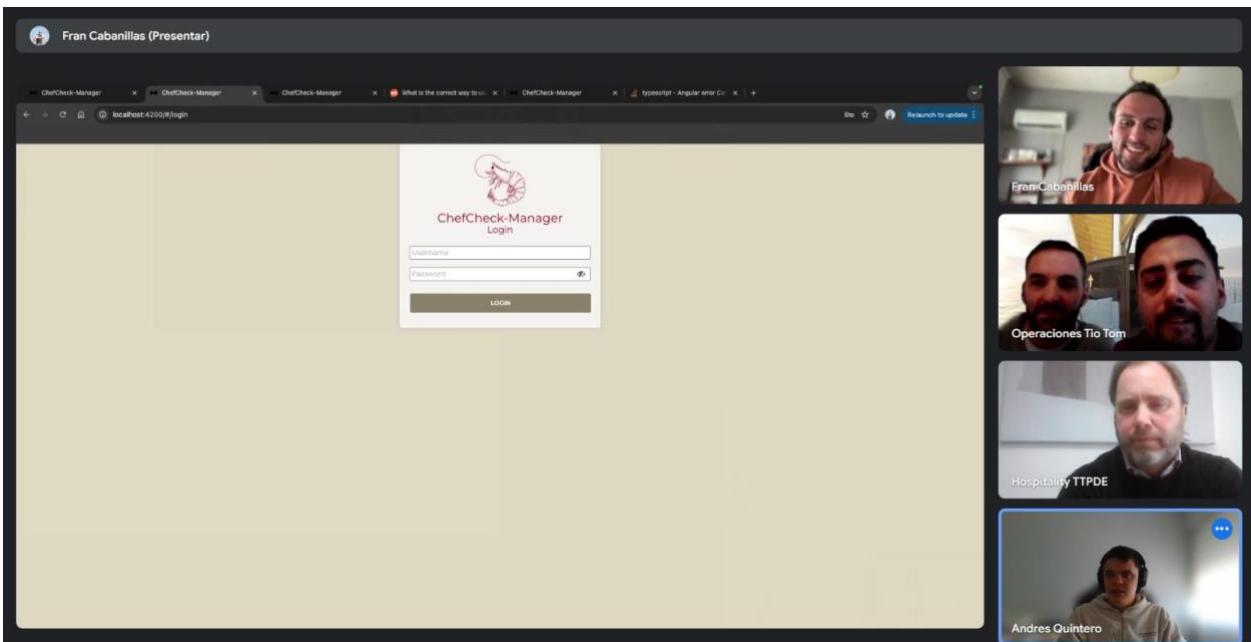


Ilustración 92 - Reunión de verificación de requerimientos con el cliente

Evidencia de reunión con Diego Dodel de FacturaLista

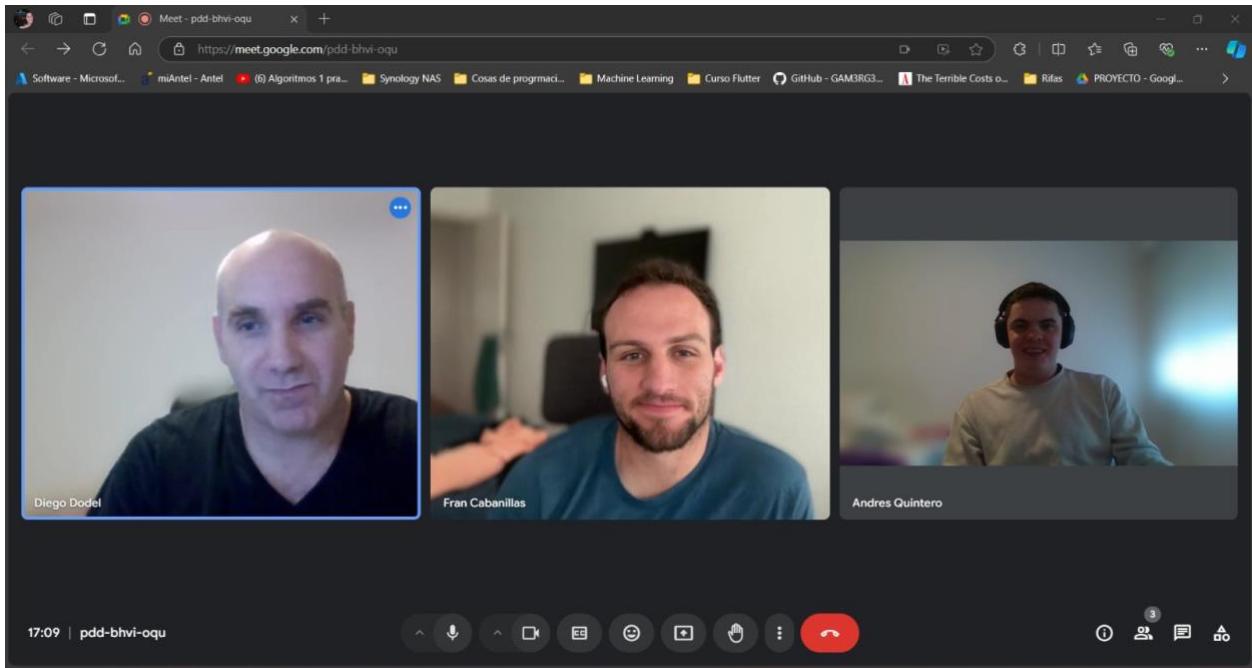


Ilustración 93 - Reunión con Diego Dodel de FacturaLista

Evidencia de Scrum daily del equipo

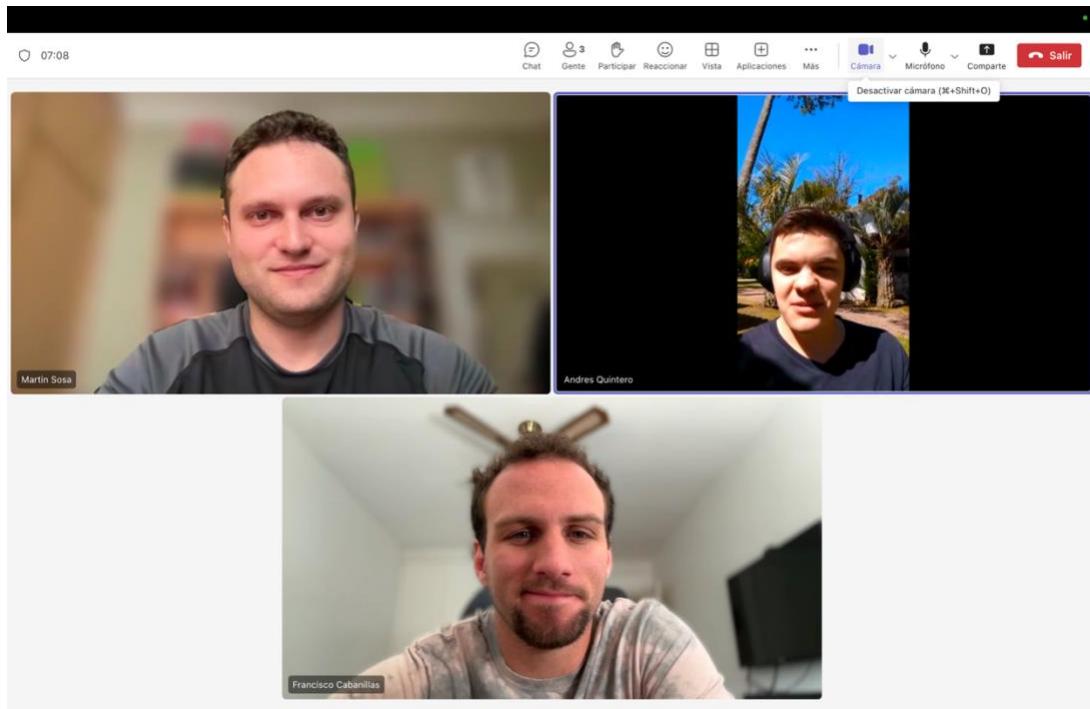


Ilustración 94 - Reunión Scrum daily del equipo

Evidencia de reunión semanal con la tutora Helena Garbarino

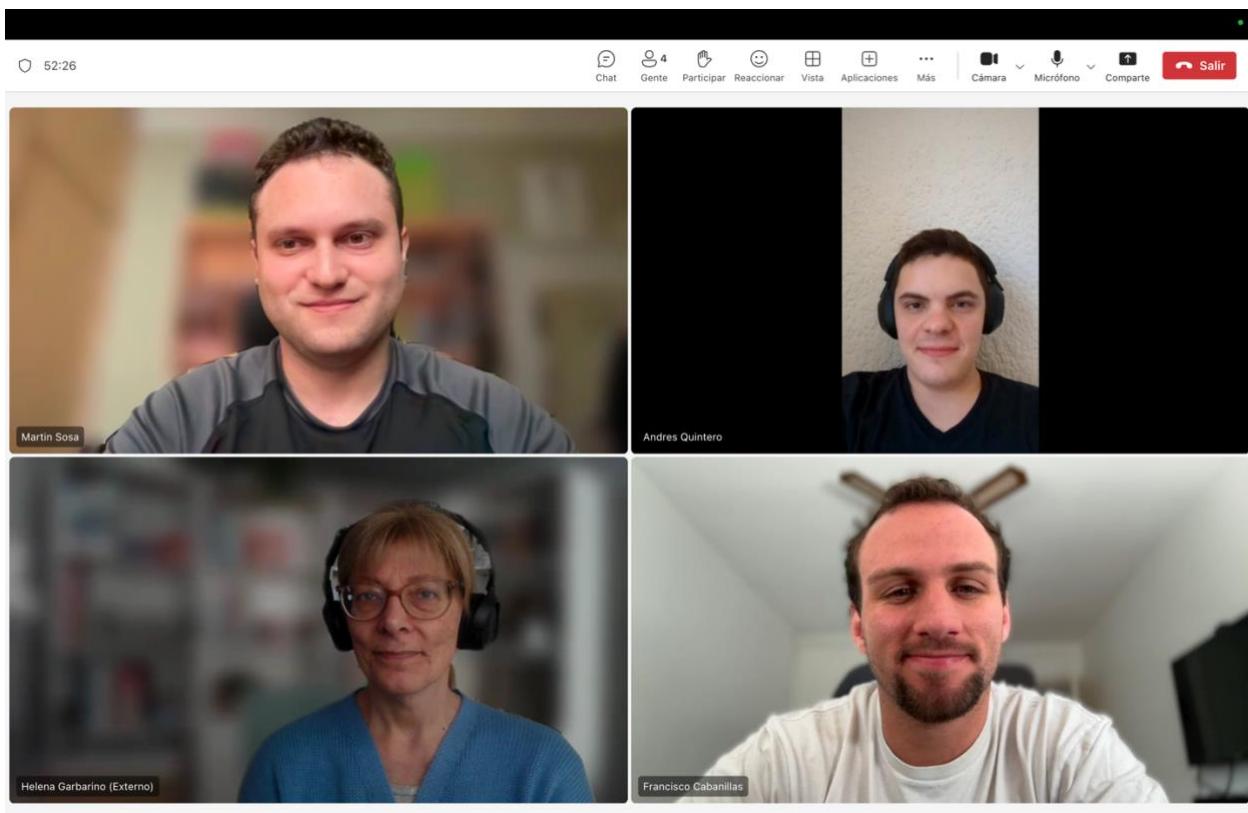


Ilustración 95 - Reunión semanal con la tutora Helena Garbarino

11.4.4. Documentación de *sprint #0* (22/4/2024 - 16/6/2024)

Planning

No hubo una planificación formal, ya que este *sprint* comprendió las actividades relacionadas a: la formalización de los requerimientos, el diseño del sistema, la elección de tecnologías, el set up de los distintos proyectos de *software*, y el comienzo de la etapa de documentación.

Al no tratarse de un *sprint* de desarrollo de *software*, en el cual se debían materializar requerimientos por medio de incrementos al finalizar un *sprint*, el equipo decidió no colocar una fecha de finalización para el mismo. El *sprint #0* culminó el día que el equipo sintió que estaban dadas las condiciones para poder comenzar a escribir código, dando paso al *sprint planning #1* (17/6/2024).

El *sprint* fue creado formalmente en Jira, donde se fueron agregando las tareas a realizar. No se realizaron estimaciones de *user stories* ya que el trabajo constaba de tareas, y no historias.

En Jira, se dejó rastro del esfuerzo dedicado, por medio de un atributo llamado "Total Effort", el cual fue creado por el equipo, donde se reflejan las horas hombre dedicadas.

Dailies

En este *sprint* especial, se realizaron reuniones semanales todos los jueves, donde se iban comentando los adelantos de los diferentes integrantes.

En el caso que un integrante se haya quedado sin tareas, se le asignaba en ese momento.

Review

El listado de tareas (Ilustración 96) realizadas en este *sprint* y su completitud se puede visualizar en la siguiente tabla (las tareas pendientes al culminar este *sprint* se trasladan automáticamente al *backlog*):

Tarea	Esfuerzo real en SP	Estado
(1) - Definición del dominio base	3	Completada
(2) - Investigar todos los aspectos de usabilidad y ergonomía contra otros productos	-	Pendiente
(3) - Gestión de calidad	14	Completada
(4) - Gestión del riesgo	4	En curso
(5) - Investigación <i>framework frontend</i>	1	Completada
(6) - Investigation <i>framework backend</i>	1	Completada
(7) - Documentación de cesión de derechos	1	Completada
(8) - Crear proyecto .NET para la <i>API</i>	2	Completada
(9) - Metodologías ágiles	3	Completada
(10) - Ciclo de vida del proyecto de <i>software</i>	3	Completada
(11) - Comparación de tecnologías de <i>Machine Learning</i>	1	Completada
(12) - Definir y configurar estructura base del proyecto <i>Backend</i>	2	Completada
(13) - Dependencias de PostgreSQL y <i>Entity</i> en la <i>API</i>	1	Completada
(14) - Crear la conexión Github + Jira	3	Completada
(15) - Configurar .net workflow (GitHub Actions)	2	Completada
(16) - Estructura de lógica de negocio y clases	1.5	Completada

(17) - Crear proyecto Flutter para el <i>frontend</i> del contenedor	1	Completada
(18) - Presupuesto Tio Tom	4	Completada
(19) - Estimar esfuerzos <i>sprint</i> 1	1	Completada
Total	48.5	18/19 Tareas completadas

Ilustración 96 - Listado de tareas *Sprint #0*

La tarea de gestión de riesgo, en realidad sí fue completada dentro de este *sprint*, pero no figura en Jira como completada, debido a que no fue actualizado su estado para el momento del cierre del *sprint*.

Burndown Chart

Al no ser un *sprint* de codificación de *software*, el equipo no consideró necesario realizar un análisis con un *burndown chart*.

Burnup Chart

Al no ser un *sprint* de codificación de *software*, el equipo no consideró necesario realizar un análisis con un *burnup chart*.

Retrospective

Al no ser un *sprint* de *software*, el equipo no consideró necesario realizar un análisis del progreso hasta el momento por medio de una retrospectiva, ya que el trabajo sucedió de manera distendida y a lo largo de un período de tiempo de casi dos meses.

11.4.5. Documentación de *sprint* #1 (17/6/2024 - 1/7/2024)

Planning

Se revisaron las historias del *backlog* y se agregaron las de mayor prioridad en el *sprint*.

El equipo repasó y estimó las historias a realizar de acuerdo con el esfuerzo recomendado por la ORT (14 hs semanales por integrante, lo cual equivale a 28 *story points* por iteración, por cada integrante). Esto es una referencia que el equipo tiene en cuenta para el esfuerzo máximo a realizar. Si es necesario un mayor esfuerzo, se puede superar esta cifra.

Uno de los objetivos de este *sprint* es poder determinar una primera versión de la velocidad promedio del equipo.

Otro muy importante objetivo (relacionado con el desarrollo en sí), es el de contar con los CRUD (a nivel de *API*) de las entidades inicialmente importantes, como, por ejemplo: Monedas, productos, tipos de productos y proveedores. De esta manera, el sistema se puede ir poblando de datos de prueba, ya que se cuenta con una estructura bastante sólida de varias de las principales entidades de negocio que utilizan el sistema.

Por último, otro objetivo de suma importancia es contar con una versión preliminar básica del sistema, la cual sirva para ser utilizado por un cliente HTTP como, por ejemplo: Swagger o Postman.

A su vez se busca tener una primera aproximación de la química del equipo, mientras se desarrollan las primeras actividades de desarrollo de *software* en su conjunto.

Daily 20/06

El integrante Martín Sosa comentó sus adelantos. Trabajó en el CRUD de las entidades referentes a los productos y tipos de productos. También implementó un control de errores en el servidor, para facilitar la captura de los mismos. Ese trabajo descrito corresponde a la *API*.NET.

Francisco comentó acerca del *refactor* realizado del CRUD anterior.

Martín se comprometió a hacer el *test* de los CRUD, una vez hayan sido completados.

Daily 22/06

El equipo trabajó sobre la documentación de cesión de derechos, donde a su vez se dejó en claro los costos y el presupuesto que tendrá que afrontar el cliente.

Daily 27/06

El integrante Martín Sosa comentó sus adelantos. Trabajó en el CRUD de la entidad referente a las monedas, así como también cuenta que desempeñó tareas de documentación del *sprint* 0. Se comprometió a agregar paginado en la *API*, lo cual ayudaría a mostrar la información de manera más organizada en el *frontend*.

Francisco Cabanillas aportó en la *API* con el CRUD de la entidad referente a los proveedores. Además, trabajó en la gestión de la autenticación, y la finalización de la documentación de cesión de derechos.

Por último, Andrés Quintero estuvo investigando acerca de la automatización de ingreso de facturas mediante la *API* de DGI.

Review

El listado de tareas (Ilustración 97) realizadas en este *sprint* y su estado de completitud se puede visualizar en la siguiente tabla (las tareas pendientes al culminar este *sprint* se trasladan automáticamente al *backlog*):

Incidencia	Tarea	User Story	Bug	Estimación en SP	Esfuerzo real en SP	Estado
(1) - Documentación de cesión de derechos	X			1	1	En revisión
(2) - Crear la conexión Github + Jira	X			1	3	Completada
(3) - Crear proyecto flutter para el <i>frontend</i> del contenedor	X			1	1	Completada
(4) - Gestión integral de proveedores		X		3	5	Completada
(5) - Automatización de la captura de datos de facturas		X		8	13	En curso
(6) - Gestión de monedas múltiples		X		2	2	Completada
(7) - Presupuesto Tío Tom	X			1.5	4	Completada
(8) - Gestión integral de productos		X		10	11	Completada
(9) - Documentación del <i>sprint</i> 0	X			1	1	Completada
(10) - Autenticación		X		9	2	En curso
(11) - Paginado en listados de todas las entidades existentes		X		2	2	Completada
Total				39.5	45	8/11 Incidencias completadas

Ilustración 97 - Listado de tareas *Sprint* #1

Burndown Chart

La siguiente gráfica muestra cómo se fueron realizando las entregas de las historias durante el *sprint*:



El detalle del gráfico (Ilustración 98) dice que el alcance del *sprint* constó de 39.5 SP, pero se completaron 21.5 SP, y no 45 como dice en la sección *review*. Eso es porque los 45 son las horas hombre, mientras que los 21.5 SP corresponden a la estimación de las tareas/*user stories* 100% completadas, sin contar el esfuerzo de tareas/*user stories* sin completar en su totalidad.

Burnup Chart

La siguiente gráfica muestra cuánto trabajo del total se ha completado:



El detalle del gráfico (Ilustración 99) dice que quedan 18 SP sin culminar. Estos 18 SP constan de la suma de los SP de estimación de aquellas tareas/*user stories* que no fueron completadas al 100%, osea, aquellas en cualquier otro estado diferente a "Completada". No importa si una *user story* tiene un estado de completitud es de 2 de 3 tareas (esto se puede interpretar como que, en el peor caso posible, el equipo tiene una deuda de 18 SP). El equipo acordó que la misma se considere completa únicamente cuando todo su contenido esté completo.

Velocidad

La diferencia entre los 39.5 de estimación de horas de trabajo hombre (*story points*), respecto a las horas completadas (45), nos deja una desviación de 4.5 *story points*.

Es decir, que el primer *sprint* marca que la velocidad total promedio del equipo en *sprints* de dos semanas, es de 45 *story points*.

El equipo está al tanto de que este número irá variando *sprint* a *sprint*.

Retrospective

El intercambio entre los integrantes del equipo se resumió en el contenido de la siguiente imagen:

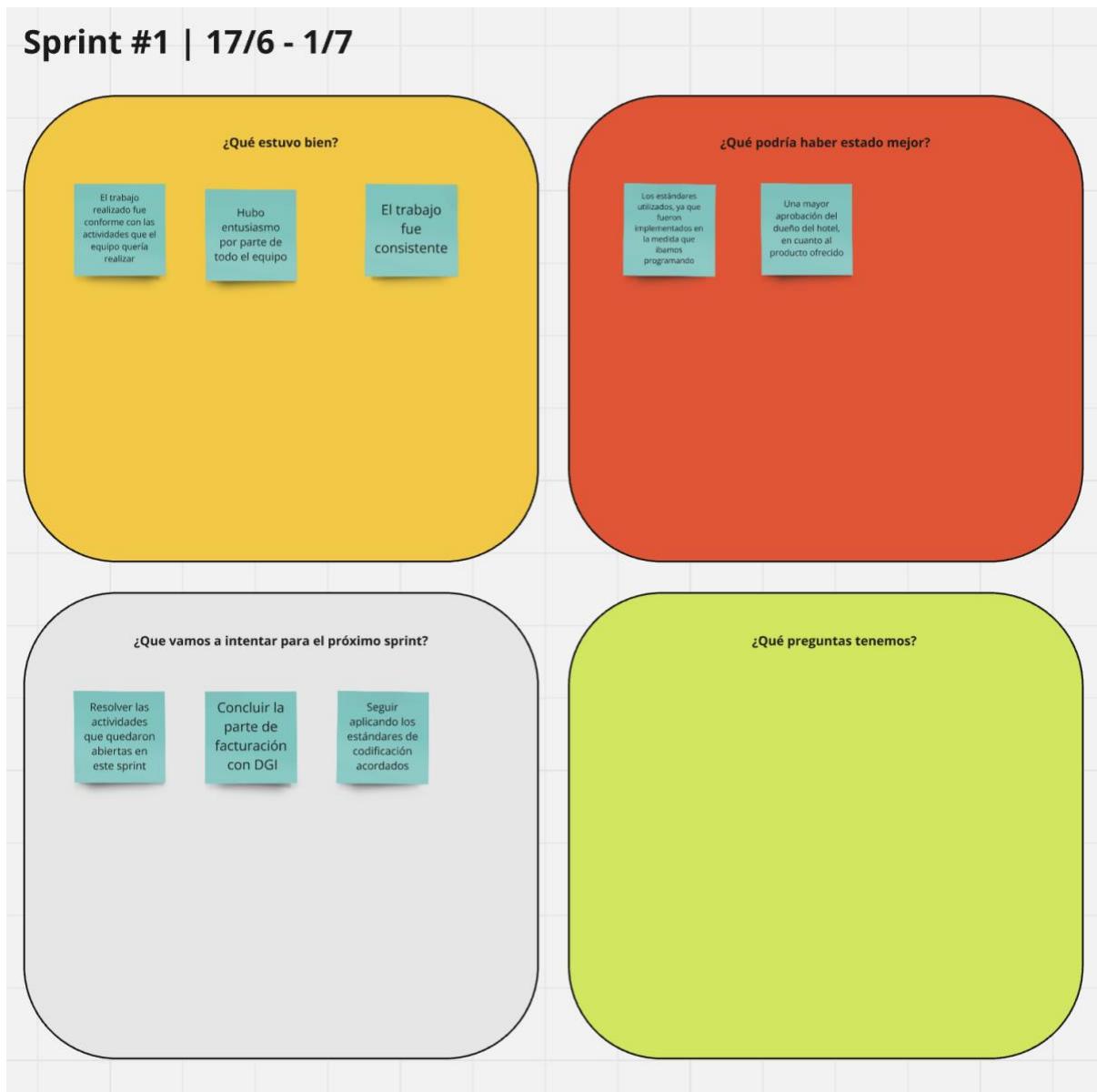


Ilustración 100- *Sprint retrospective* del *sprint #1*

11.4.6. Documentación de *sprint* #2 (1/7/2024 - 15/7/2024)

Planning

Se revisaron las historias del *backlog* y se agregaron las de mayor prioridad en el *sprint*.

El equipo repasó y estimó las historias a realizar de acuerdo con el esfuerzo recomendado por la ORT (14 hs semanales por integrante, lo cual equivale a 28 *story points* por iteración, por cada integrante). Esto es una referencia que el equipo tiene en cuenta para el esfuerzo máximo a realizar. Si es necesario un mayor esfuerzo, se puede superar esta cifra.

El objetivo del *sprint* es tener desarrollado del lado del servidor todos los CRUD necesarios para poder consumir la *API* desde Swagger o POSTMAN, para poder crear, modificar, eliminar y obtener las entidades más importantes del sistema.

Otro objetivo importante es generar avances en lo que corresponde a la obtención de facturas desde uno o varios *endpoints* de DGI.

Daily 4/7

Martín Sosa trabajó en *user stories* relacionadas con el ingreso y egreso de mercadería del contenedor.

Francisco Cabanillas trabajó en la autenticación a nivel de la *API*.

Andrés Quintero nos comentó sobre las averiguaciones con DGI, para la implementación del data entry de facturas automático.

Se comienza a trabajar en la presentación visual para la primera revisión del equipo.

Daily 11/7

El equipo no mostró un progreso significativo debido a que dos de los integrantes tienen el examen de la materia Arquitecturas Empresariales, y el tercer integrante tuvo una entrega del obligatorio de Analítica para Datawarehousing.

Andrés Quintero se comprometió a llamar a al menos dos proveedores de factura electrónica para seguir estudiando la posibilidad de realizar una integración directamente con la *API* de DGI, o mediante la utilización de un servicio ya existente por parte de alguno de estos proveedores.

Review

El listado de tareas (Ilustración 101) realizadas en este *sprint* y su estado de completitud se puede visualizar en la siguiente tabla (las tareas pendientes al culminar este *sprint* se trasladan automáticamente al *backlog*):

Incidencia	Tarea	User Story	Bug	Estimación en SP	Esfuerzo real en SP	Estado
(1) - Manejo de valor nulo en GetObjectById	X			1	-	Pendiente
(2) - Documentación del <i>sprint</i> 1	X			2	2	Completada
(3) - Registro de ingreso de mercadería		X		1.5	1.5	Completada
(4) - Registro de egreso de mercadería		X		1.5	1.5	Completada
(5) - Autenticación <i>web</i>		X		9	7	En curso
(6) - Documentación de cesión de derechos	X			1	1	Completada
(7) - Automatización de la captura de datos de facturas		X		8	13	En curso
(8) - Visualización de facturas por proveedor		X		2	-	Pendiente
(9) - Presentación para la revisión	X			5	-	En curso
(10) - Ingreso manual de datos de facturas		X		10	-	Pendiente
(11) - Eliminación de movimiento de inventario		X		1.5	1.5	Completada
(12) - Visualización de egresos		X		1.5	1.5	Completada
(13) - Documentación del <i>sprint</i> 2	X			2	1	En curso
Total				46	30	6/13 Incidencias completadas

Ilustración 101 - Listado de tareas *Sprint* #2

Burndown Chart

La siguiente gráfica muestra cómo se fueron realizando las entregas de las historias durante el *sprint*:

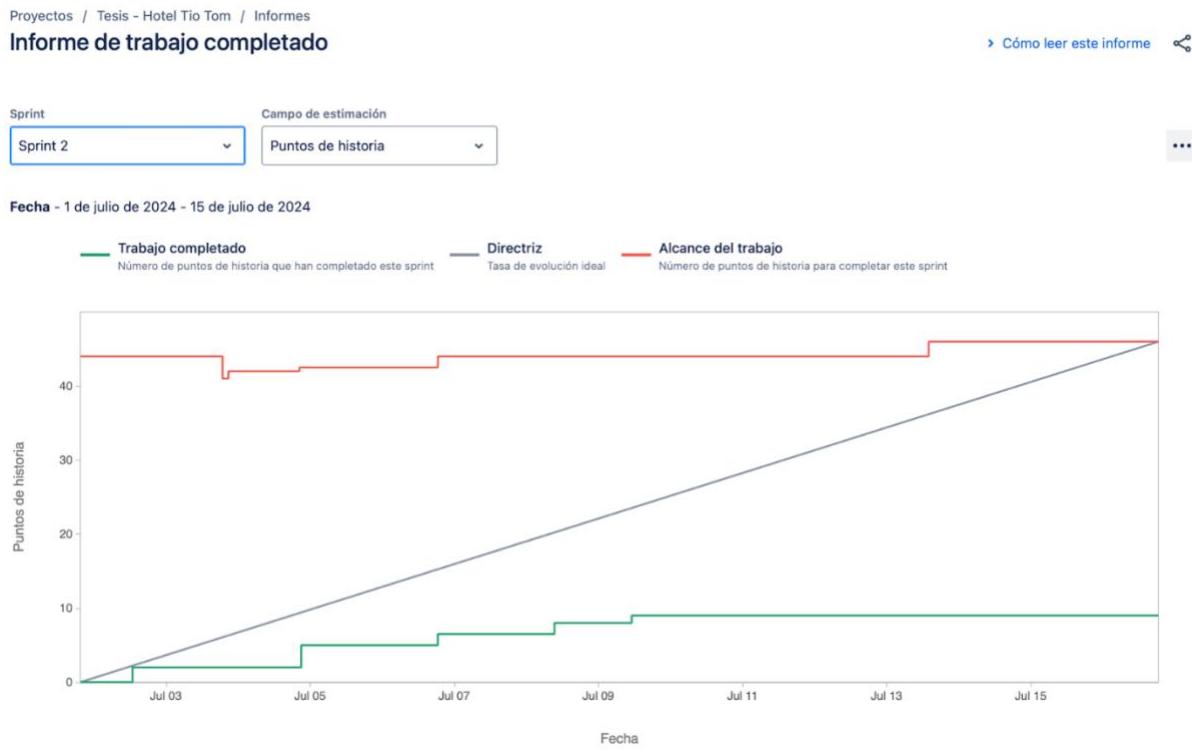


Ilustración 102 - *Burndown chart* del *sprint* #2

El detalle del gráfico (Ilustración 102) dice que el alcance del *sprint* constó de 46 SP, pero se completaron 9 SP, y no 30 como dice en la sección *review*. Eso es porque los 30 son las horas hombre, mientras que los 9 SP corresponden a la estimación de las tareas/*user stories* 100% completadas, sin contar el esfuerzo de tareas/*user stories* sin completar en su totalidad.

Burnup Chart

La siguiente gráfica muestra cuánto trabajo del total se ha completado:



Ilustración 103 - *Burnup chart del sprint #2*

El detalle del gráfico (Ilustración 103) dice que quedan 37 SP sin culminar. Estos 37 SP constan de la suma de los SP de estimación de aquellas tareas/*user stories* que no fueron completadas al 100%, osea, aquellas en cualquier otro estado diferente a "Completada". No importa si una *user story* tiene un estado de completitud es de 2 de 3 tareas (esto se puede interpretar como que, en el peor caso posible, el equipo tiene una deuda de 37 SP). El equipo acordó que la misma se completará cuando todo su contenido esté completo.

Velocidad

La diferencia entre los 46 de estimación de horas de trabajo hombre (*story points*), respecto a las horas completadas (30), nos deja una desviación de 16 *story points*.

Es decir, que el segundo *sprint* marca que la velocidad total promedio del equipo es de: (Velocidad *Sprint#1* (45) + Velocidad *Sprint#2* (30)) / Cantidad de *Sprints* (2) = 37.5 *story points*.

El equipo está al tanto de que este número irá variando *sprint* a *sprint*.

Retrospective

El intercambio entre los integrantes del equipo se resumió en el contenido de la siguiente imagen:

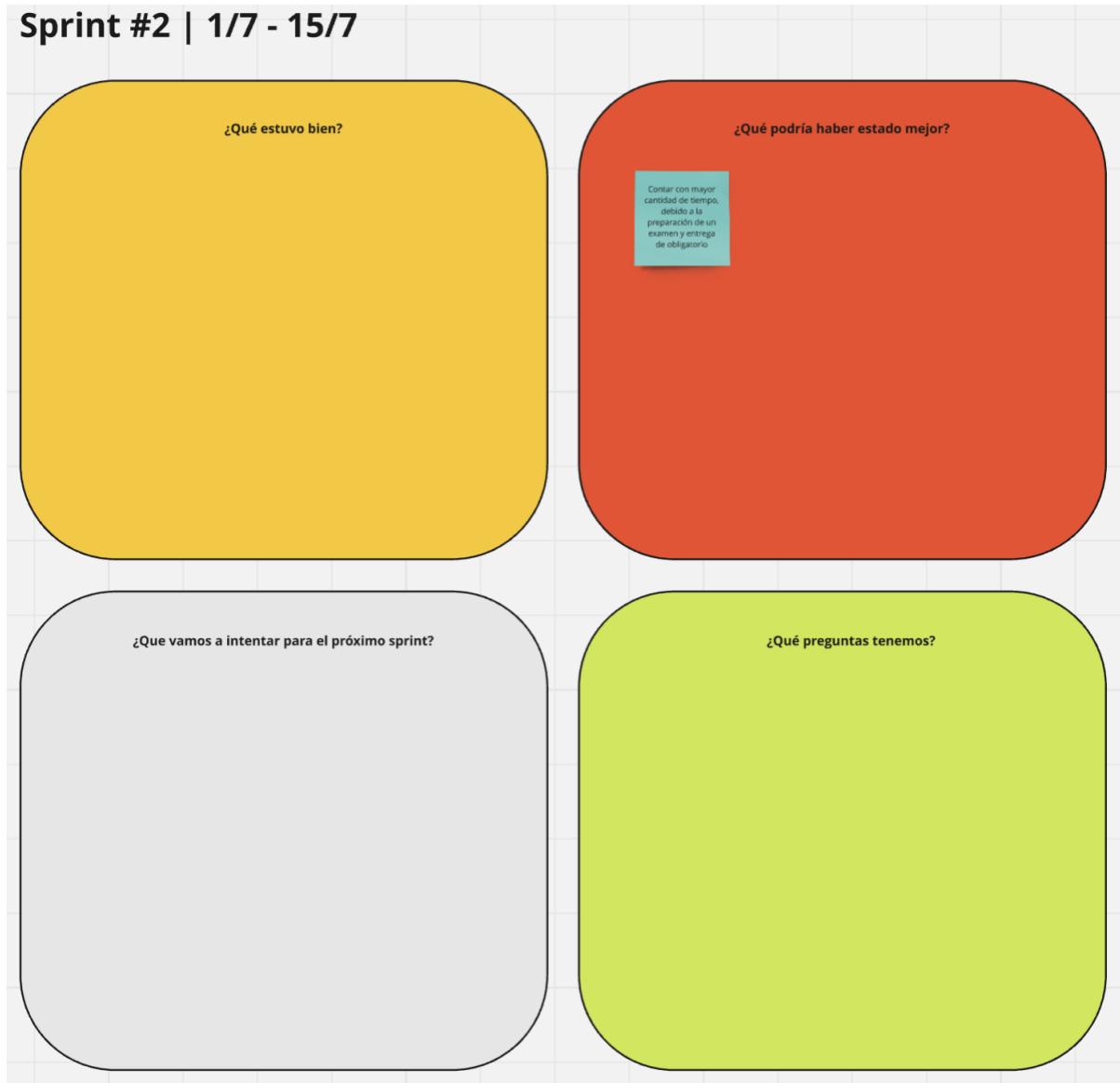


Ilustración 104 - *Sprint retrospective* del *sprint* #2

11.4.7. Documentación de *sprint* #3 (15/7/2024 - 29/7/2024)

Planning

Se revisaron las historias del *backlog* y se agregaron las de mayor prioridad en el *sprint*.

El equipo repasó y estimó las historias a realizar de acuerdo con el esfuerzo recomendado por la ORT (14 hs semanales por integrante, lo cual equivale a 28 *story points* por iteración, por cada integrante). Esto es una referencia que el equipo tiene en cuenta para el esfuerzo máximo a realizar. Si es necesario un mayor esfuerzo, se puede superar esta cifra.

El objetivo del *sprint* es comenzar a materializar en código los aspectos relacionados a la facturación y DGI (crear, listar, conectar con *APIs* de terceros, etc).

Otro objetivo importante es poder comenzar a darle forma a la aplicación móvil Flutter, con casos de uso iniciales y básicos como, por ejemplo: *splash screen*, *login*, *logout* y menú.

El equipo tiene la intención de poder culminar toda la parte de *API* necesaria para soportar los casos de uso del punto anterior.

Daily 18/7

Martín Sosa se comprometió a trabajar en el *endpoint* que devuelve el listado de facturas, el cual tiene determinados filtros, que el usuario decide o no utilizar. A su vez también se comprometió a avanzar con la gestión de logs cuando ocurre una excepción.

Francisco Cabanillas y Andrés Quintero se encuentran en la semana previa al examen de Arquitecturas Empresariales, y, por ende, su participación en el *sprint* se ve limitada hasta pasado el examen del 23 de julio.

Daily 22/7

Martín Sosa se comprometió a avanzar con la gestión de logs cuando ocurre una excepción.

Andrés Quintero nos cuenta que desde uno de los proveedores de factura electrónica al cual se contactó, no le han respondido su email.

No se cuentan con avances en código de la sección de facturación con DGI o un proveedor de facturación (únicamente existe el *endpoint* de Martín Sosa, descrito en la *daily* del 18/7).

El equipo tiene un asunto urgente que atender, el cual es la presentación a dar en la primera revisión (29/7), la cual consta de una presentación presencial. Ese será todo el trabajo del resto del *sprint*.

Review

El listado de tareas (Ilustración 105) realizadas en este *sprint* y su estado de completitud se puede visualizar en la siguiente tabla (las tareas pendientes al culminar este *sprint* se trasladan automáticamente al *backlog*):

Incidencia	Tarea	User Story	Bug	Estimación en SP	Esfuerzo real en SP	Estado
(1) Manejo de valor nulo en GetObjectById	X			1	-	Pendiente
(2) Autenticación web		X		9	-	En curso
(3) Automatización de la captura de datos de facturas		X		8	-	En curso
(4) Ingreso manual de datos de facturas		X		10	-	Pendiente
(5) Presentación para la revisión	X			5	5	Completada
(6) Documentación del <i>sprint</i> 2	X			2	2	Completada
(7) Configuración proyecto Flutter	X			2	-	Pendiente
(8) Búsqueda, orden y filtrado de facturas		X		3	3	Completada
(9) Crear proyecto Angular para el <i>frontend</i> de la web	X			-	-	Pendiente (no se llegó a estimar)
(10) Bug en <i>login endpoint</i>			X	-	-	Pendiente (no se llegó a estimar)
(11) Agregar log al ExceptionFilter	X			4	5	Completada
Total				44	15	4/11 Incidencias completadas

Ilustración 105 - Listado de tareas *Sprint* #3

Burndown Chart

La siguiente gráfica muestra cómo se fueron realizando las entregas de las historias durante el *sprint*:



Ilustración 106 - *Burndown chart* del *sprint* #3

El detalle del gráfico (Ilustración 106) dice que el alcance del *sprint* constó de 44 SP, pero se completaron 9 SP, y no 15 como dice en la sección *review*. Eso es por dos razones: 1) Porque los 15 son las horas hombre, mientras que los 9 SP corresponden a la estimación de las tareas/*user stories* 100% completadas, sin contar el esfuerzo de tareas/*user stories* sin completar en su totalidad. 2) Además, al cerrar el *sprint*, el equipo se olvidó de marcar como terminada algunas tareas, y por lo tanto los datos del *sprint* culminado no varían. Por eso el equipo desea aclarar que de los 44 SP acordados, realmente se cumplieron 15 y no 9 como indica Jira.

Burnup Chart

La siguiente gráfica muestra cuánto trabajo del total se ha completado:

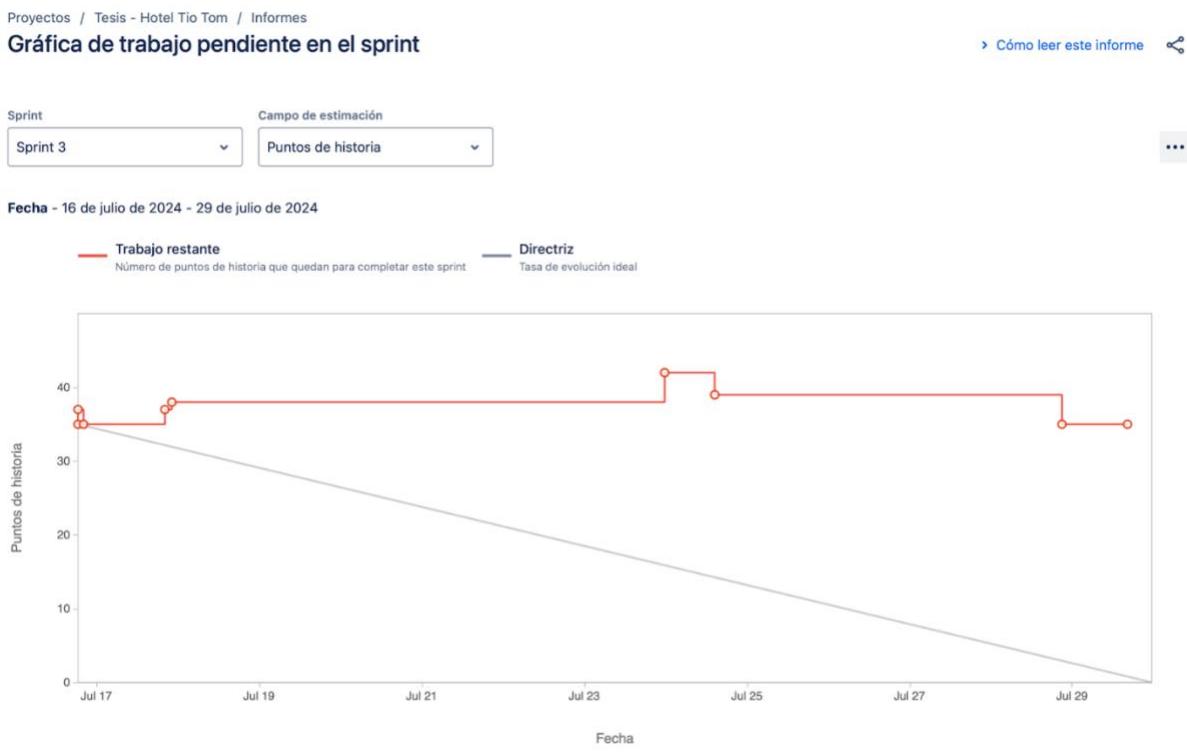


Ilustración 107 - Burnup chart del sprint #3

El detalle del gráfico (Ilustración 107) dice que quedan 35 SP sin culminar. Estos 35 SP constan de la suma de los SP de estimación de aquellas tareas/*user stories* que no fueron completadas al 100%, osea, aquellas en cualquier otro estado diferente a "Completada". No importa si una *user story* tiene un estado de completitud es de 2 de 3 tareas (esto se puede interpretar como que, en el peor caso posible, el equipo tiene una deuda de 35 SP). El equipo acordó que la misma se completará cuando todo su contenido esté completo.

Velocidad

La diferencia entre los 44 de estimación de horas de trabajo hombre (*story points*), respecto a las horas completadas (15), nos deja una desviación de 29 *story points*.

Es decir, que el tercer *sprint* marca que la velocidad total promedio del equipo es de: (Velocidad *Sprint#1* (45) + Velocidad *Sprint#2* (30) + Velocidad *Sprint#3* (15)) / Cantidad de *Sprints* (3) = 30 *story points*.

El equipo está al tanto de que este número irá variando *sprint* a *sprint*.

Retrospective

El intercambio entre los integrantes del equipo se resumió en el contenido de la siguiente imagen:

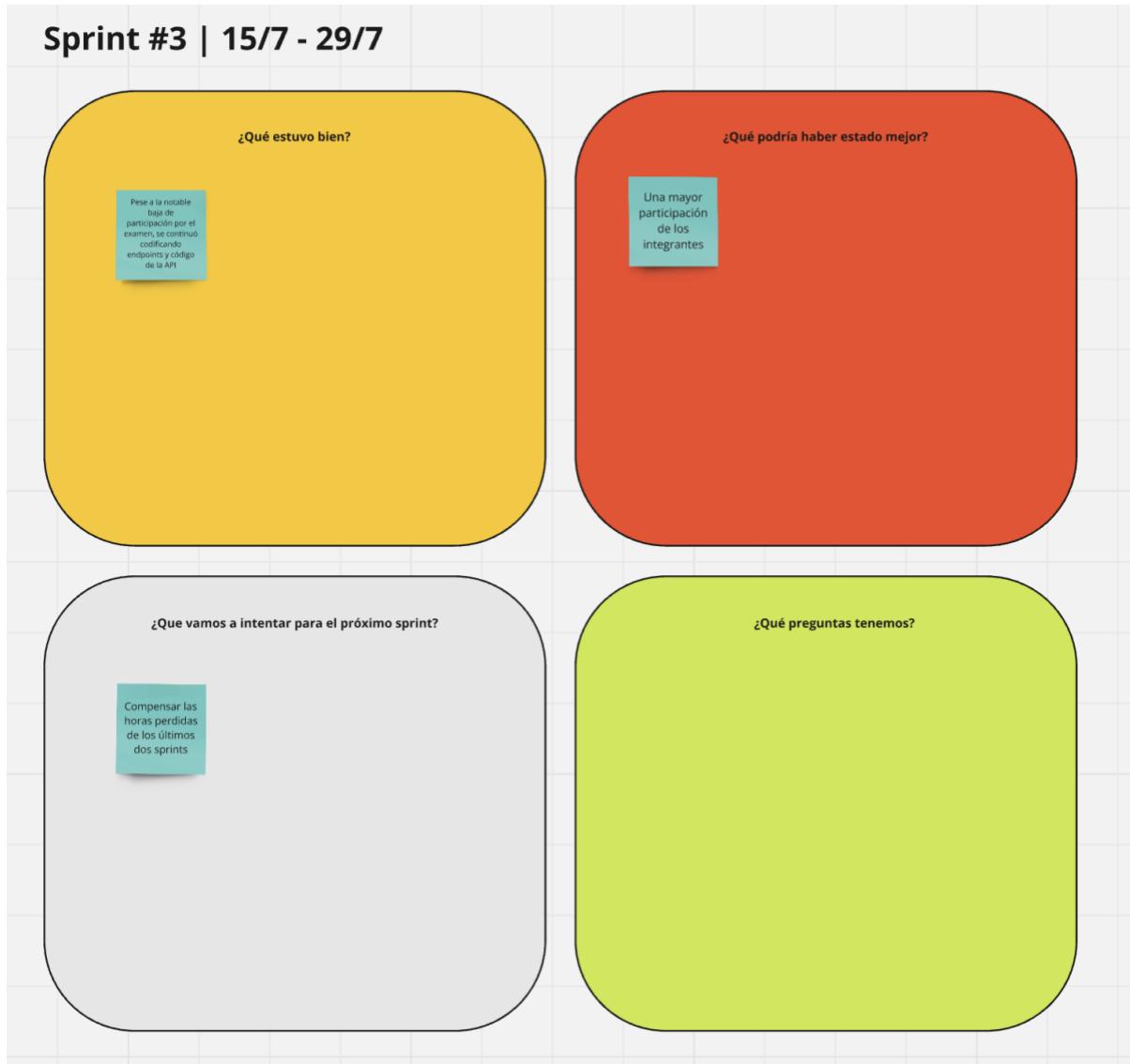


Ilustración 108 - *Sprint retrospective* del *sprint* #3

11.4.8. Documentación de *sprint* #4 (29/7/2024 - 12/8/2024)

Planning

Se revisaron las historias del *backlog* y se agregaron las de mayor prioridad en el *sprint*.

El equipo repasó y estimó las historias a realizar de acuerdo con el esfuerzo recomendado por la ORT (14 hs semanales por integrante, lo cual equivale a 28 *story points* por iteración, por cada integrante). Esto es una referencia que el equipo tiene en cuenta para el esfuerzo máximo a realizar. Si es necesario un mayor esfuerzo, se puede superar esta cifra.

Los objetivos del *sprint* son: 1) Contar con una aproximación visual de la aplicación Flutter del contenedor (autenticación, listado de los movimientos de inventario y eliminación de los mismos). 2) Configurar el proyecto *web* Angular, junto al manejo de autenticación. 3) Gestión manual de las facturas (sin integración con DGI o FacturaLista). 4) Documentar sobre arquitectura y requerimientos no funcionales (gracias al *feedback* de la primera revisión).

Daily 1/8

Martín Sosa compartió sus resultados. Realizó la documentación completa de la primera versión de la arquitectura. Además, creó un diagrama con la línea temporal del proyecto, desde la entrega del anteproyecto, hasta la entrega del producto al cliente.

Daily 5/8

Martín Sosa continuó avanzando con la documentación. Realizó todos los diagramas correspondientes de la arquitectura (los cuales, dado la fecha, son modificables ya que el desarrollo se encuentra lejos de culminar). Se comprometió (para la otra semana del *sprint*), comenzar con la aplicación móvil Flutter del contenedor.

Daily 11/8

Martín Sosa cuenta los avances de la aplicación móvil. Logró programar el *splash screen*, los íconos, el *login*, *logout* y el listado de los movimientos del inventario.

Andrés Quintero cuenta que hay que realizar un *refactor* del código fuente de la *API* (el cual es bien visto por los otros dos integrantes del equipo). A su vez cuenta que estuvo trabajando en la gestión de facturas.

Francisco cuenta que terminó la gestión de la autenticación en la *API* (roles y permisos para los usuarios). Ahora mismo se encuentra configurando el proyecto de Angular *web*, para poder comenzar con la autenticación.

Review

El listado de tareas (Ilustración 109) realizadas en este *sprint* y su estado de completitud se puede visualizar en la siguiente tabla (las tareas pendientes al culminar este *sprint* se trasladan automáticamente al *backlog*):

Incidencia	Tarea	User Story	Bug	Estimación en SP	Esfuerzo real en SP	Estado
(1) Automatización de la captura de datos de facturas		X		8	-	En curso
(2) Bug en <i>login endpoint</i>			X	1	1	Completada
(3) Configuración proyecto Flutter	X			1	1	Completada
(4) Manejo de valor nulo en <i>GetObjectById</i>	X			1	-	Pendiente
(5) Configuración proyecto Angular	X			2	3	Completada
(6) Autenticación <i>web</i>		X		9	11	Completada
(7) Ingreso manual de datos de facturas		X		10	12	Completada
(8) Presentación para la revisión 1	X			5	5	Completada
(9) Requerimientos no funcionales	X			1	1	Completada
(10) Versión inicial de la Arquitectura del sistema	X			9	11	Completada
(11) Atributos de calidad en base a requerimientos no funcionales	X			4	-	Pendiente
(12) Definición de desafíos	X			4	-	Pendiente
(13) Línea de tiempo de <i>sprints</i> , cronograma	X			2	2	Completada
(14) Revisión técnica	X			1	1	Completada
(15) Informe de la revisión	X			1	1	Completada
(16) Documentación <i>sprint 3</i>	X			1	1	Completada
(17) Solicitar y agendar revisión técnica	X			1	1	Completada
(18) Autenticación <i>mobile</i>		X		4	7	Completada
(19) <i>Home Page</i> y navegación a diferentes pantallas <i>app flutter</i>	X			5	3	Completada
(20) <i>Login web</i>		X		4	-	En curso

(21) Gestión/asignación de roles y permisos		X		5	-	Pendiente
(22) Listar movimientos de inventario		X		5	5	Completada
(23) Refactor API .NET	X			10	-	En curso
(24) Agregar nuevo movimiento de inventario		X		4	4	Completada
Total				98	70	17/24 Incidencias completadas

Ilustración 109 - Listado de tareas **Sprint #4**

Burndown Chart

La siguiente gráfica muestra cómo se fueron realizando las entregas de las historias durante el *sprint*:

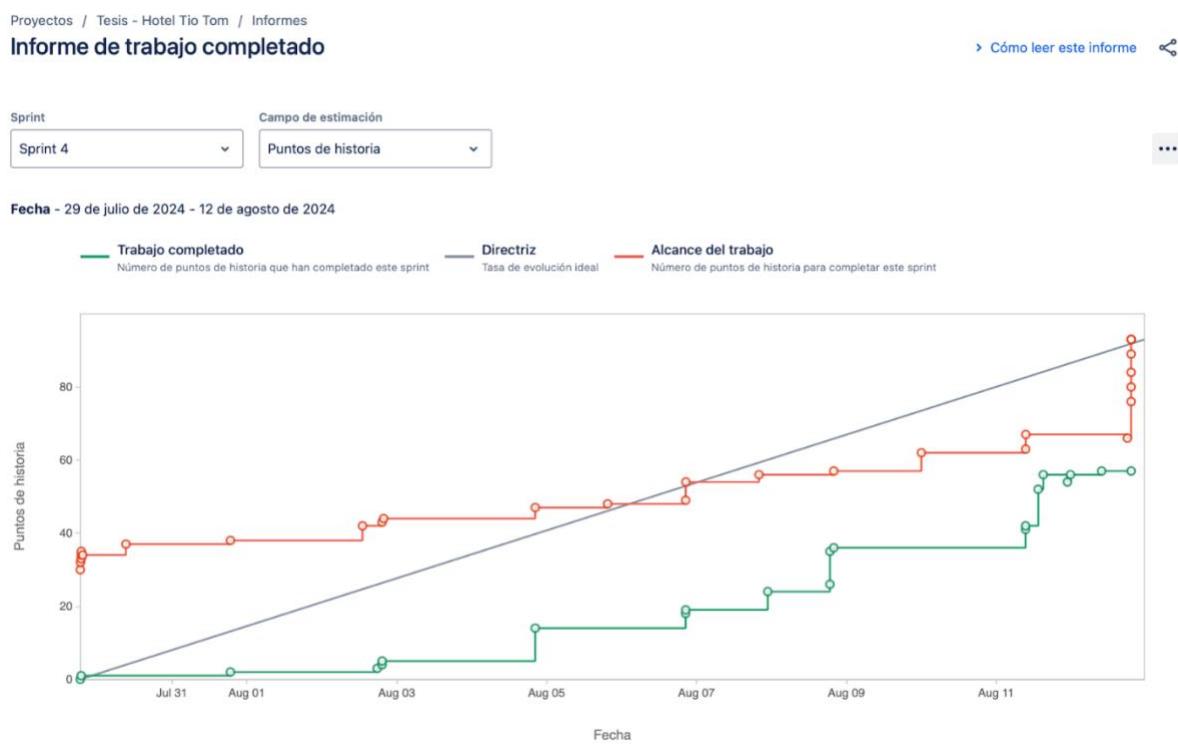


Ilustración 110 - *Burndown chart* del *sprint #4*

El detalle del gráfico (Ilustración 110) dice que el alcance del *sprint* constó de 93 SP, pero se completaron 57 SP, y no 70 como dice en la sección *review*. Eso es porque los 70 son las horas hombre, mientras que los 57 SP corresponden a la estimación de las tareas/*user stories* 100%

completadas, sin contar el esfuerzo de tareas/*user stories* sin completar en su totalidad. Por otro lado, no son 93 SP los comprometidos por el equipo (tal como muestra Jira), sino que fueron 98, solo que, por error, no quedó almacenada una estimación.

Burnup Chart

La siguiente gráfica muestra cuánto trabajo del total se ha completado:



Ilustración 111 - Burnup chart del sprint #4

El detalle del gráfico (Ilustración 111) dice que quedan 36 SP sin culminar. Estos 36 SP constan de la suma de los SP de estimación de aquellas tareas/*user stories* que no fueron completadas al 100%, osea, aquellas en cualquier otro estado diferente a "Completada". No importa si una *user story* tiene un estado de completitud es de 2 de 3 tareas (esto se puede interpretar como que, en el peor caso posible, el equipo tiene una deuda de 36 SP). El equipo acordó que la misma se completará cuando todo su contenido esté completo.

Velocidad

La diferencia entre los 98 de estimación de horas de trabajo hombre (*story points*), respecto a las horas completadas (70), nos deja una desviación de 28 *story points*.

Es decir, que el cuarto *sprint* marca que la velocidad total promedio del equipo es de: (Velocidad *Sprint#1* (45) + Velocidad *Sprint#2* (30) + Velocidad *Sprint#3* (15) + Velocidad *Sprint#4* (70)) / Cantidad de *Sprints* (4) = 40 *story points*.

El equipo está al tanto de que este número irá variando *sprint* a *sprint*.

Retrospective

El intercambio entre los integrantes del equipo se resumió en el contenido de la siguiente imagen:

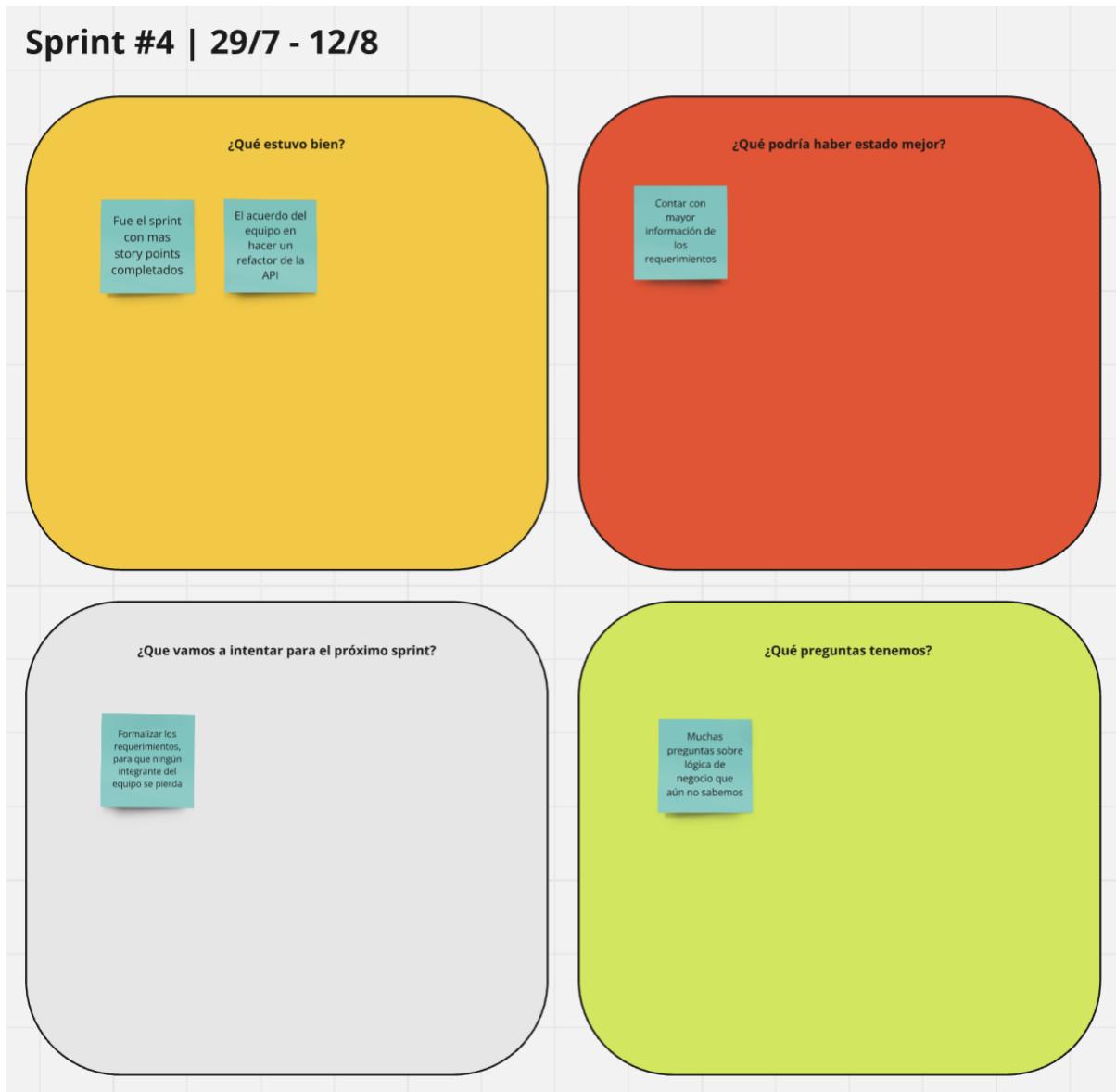


Ilustración 112 - *Sprint retrospective* del *sprint #4*

11.4.9. Documentación de *sprint* #5 (12/8/2024 - 26/8/2024)

Planning

Se revisaron las historias del *backlog* y se agregaron las de mayor prioridad en el *sprint*.

El equipo repasó y estimó las historias a realizar de acuerdo con el esfuerzo recomendado por la ORT (14 hs semanales por integrante, lo cual equivale a 28 *story points* por iteración, por cada integrante). Esto es una referencia que el equipo tiene en cuenta para el esfuerzo máximo a realizar. Si es necesario un mayor esfuerzo, se puede superar esta cifra.

Los objetivos del *sprint* son: 1) Tener una primera versión de la aplicación que utilizarán los operarios en el contenedor de alimentos. 2) Hacer un *refactor* de la *API*, teniendo en cuenta las reglas de negocio definidas. 3) Una documentación formal de ingeniería de requerimientos. 4) En base al resultado de la revisión técnica, seguir mejorando la documentación de la arquitectura, para lograr una primera versión completa de la misma

Reunión con cliente 14/8

El equipo se reunió con el cliente para comentar y mostrar los adelantos visuales tanto de la aplicación *web* como la móvil.

El cliente mostró conformidad con lo logrado.

El cliente hizo hincapié en un requerimiento importante (aún no implementado, pero ya conocido), el cual consta de dejar registro de la cantidad de comensales por turno (desayuno, almuerzo, merienda y cena). De esta manera, ellos pueden conocer, por ejemplo, cuántos comensales hubo en el almuerzo del 2 de enero del 2024.

Daily 16/8

Andrés Quintero comentó los avances del *refactor* de la *API*.

Martín cuenta que tiene una primera versión de la aplicación móvil Flutter casi pronta. Queda a la espera del *feedback* con el cliente.

Francisco Cabanillas estuvo trabajando en el *login* y ruteo del panel *web* admin Angular.

Daily 22/8

Martín se encuentra trabajando en el *refactor* de la *app* móvil, para que a raíz de los cambios del *refactor* de Andrés con la *API*, no haya errores.

Andrés sigue trabajando en el *refactor* de la *API*.

Francisco tiene preparado el CRUD de la entidad correspondiente a los proveedores, en la *web* Angular.

Review

El listado de tareas (Ilustración 113) realizadas en este *sprint* y su estado de completitud se puede visualizar en la siguiente tabla (las tareas pendientes al culminar este *sprint* se trasladan automáticamente al *backlog*):

Incidencia	Tarea	User Story	Bug	Estimación en SP	Esfuerzo real en SP	Estado
(1) Login web		X		4	10	Completada
(2) Atributos de calidad en base a requerimientos no funcionales	X			4	-	Pendiente
(3) Automatización de la captura de datos de facturas		X		8	13	En curso
(4) Refactor API .NET	X			10	16	En revisión
(5) Eliminación de movimiento de inventario		X		2	3	Completada
(6) Modificar documentación de Arquitectura pos revisión técnica	X			2	4	Completada
(7) Refactor Flutter	X			2	7	En curso
(8) Informe de avance	X			5	5	Completado
(9) Ingeniería de requerimientos	X			5	-	Pendiente
(10) Documentación del <i>sprint</i> 4	X			2	2	Completada
(11) Gestión de proveedores		X		5	4	En curso
Total				49	64	5/11 Incidencias completadas

Ilustración 113 - Listado de tareas Sprint #5

Burndown Chart

La siguiente gráfica muestra cómo se fueron realizando las entregas de las historias durante el *sprint*:

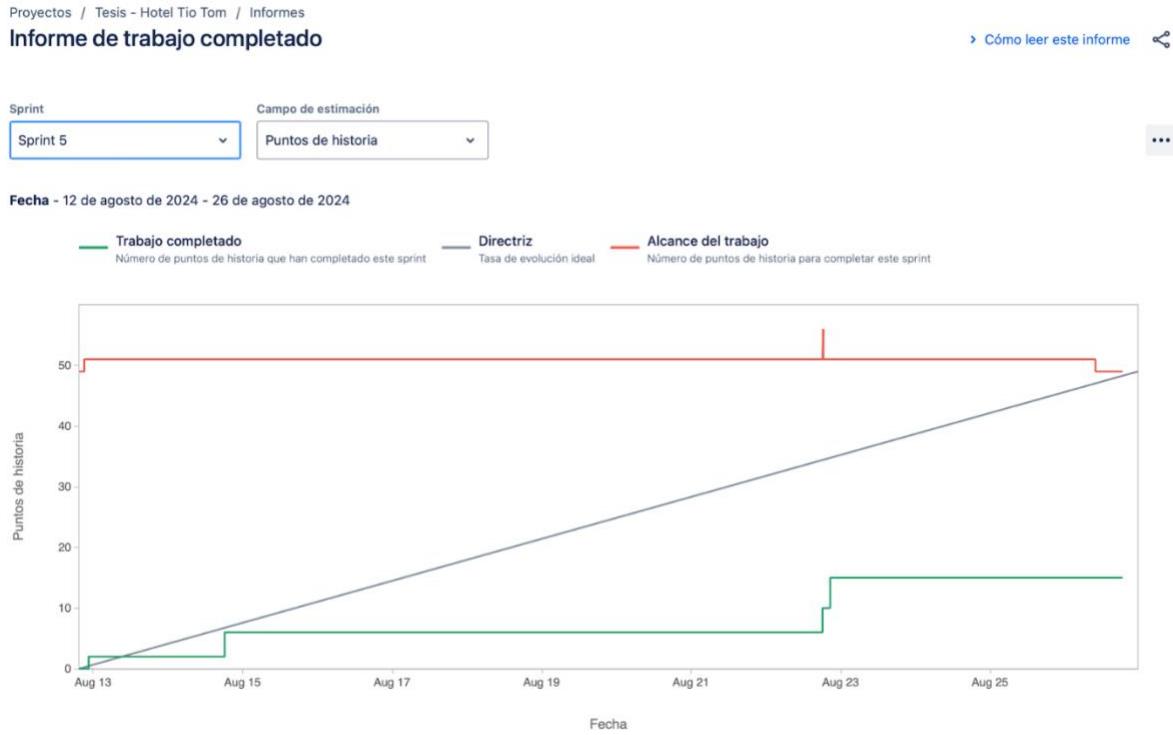


Ilustración 114 - *Burndown chart* del *sprint* #5

El detalle del gráfico (Ilustración 114) dice que el alcance del *sprint* constó de 49 SP, pero se completaron 15 SP, y no 64 como dice en la sección *review*. Eso es porque los 64 son las horas hombre, mientras que los 15 SP corresponden a la estimación de las tareas/*user stories* 100% completadas, sin contar el esfuerzo de tareas/*user stories* sin completar en su totalidad.

Burnup Chart

La siguiente gráfica muestra cuánto trabajo del total se ha completado:

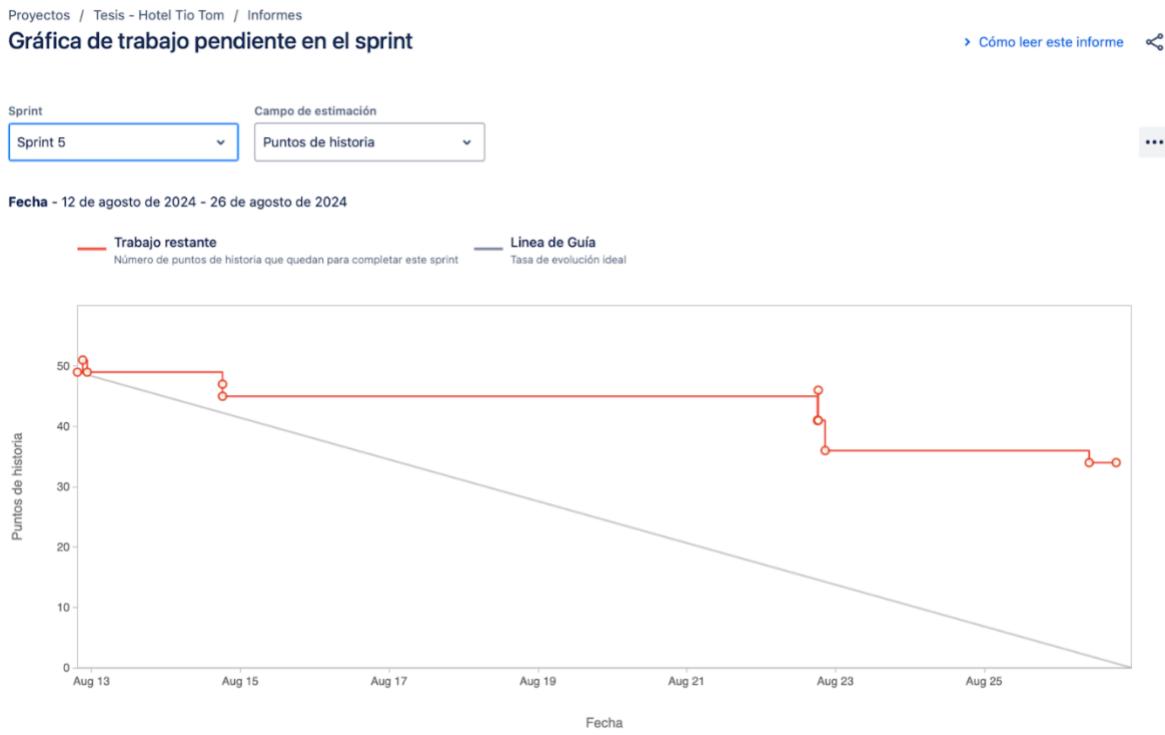


Ilustración 115 - Burnup chart del sprint #5

El detalle del gráfico (Ilustración 115) dice que quedan 34 SP sin culminar. Estos 34 SP constan de la suma de los SP de estimación de aquellas tareas/*user stories* que no fueron completadas al 100%, osea, aquellas en cualquier otro estado diferente a "Completada". No importa si una *user story* tiene un estado de completitud es de 2 de 3 tareas. El equipo acordó que la misma se completará cuando todo su contenido esté completo.

Velocidad

La diferencia entre los 49 de estimación de horas de trabajo hombre (*story points*), respecto a las horas completadas (64), nos deja una desviación de 15 *story points*.

Es decir, que el quinto *sprint* marca que la velocidad total promedio del equipo es de: (Velocidad *Sprint#1* (45) + Velocidad *Sprint#2* (30) + Velocidad *Sprint#3* (15) + Velocidad *Sprint#4* (70) + Velocidad *Sprint#5* (64)) / Cantidad de *Sprints* (5) = 44.8 *story points*.

El equipo está al tanto de que este número irá variando *sprint* a *sprint*.

Retrospective

El intercambio entre los integrantes del equipo se resumió en el contenido de la siguiente imagen:

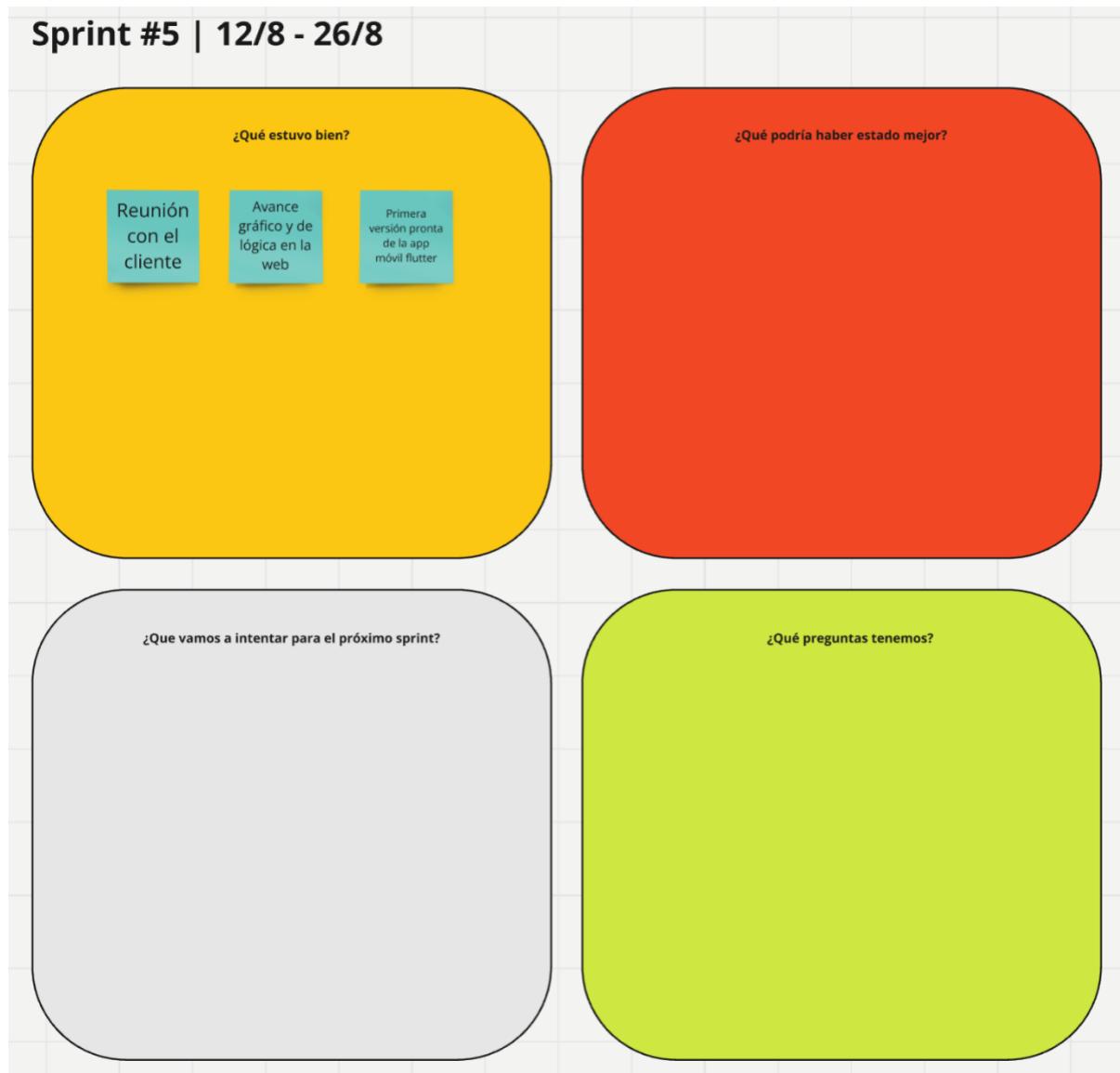


Ilustración 116 - *Sprint retrospective* del *sprint #5*

11.4.10. Documentación de *sprint* #6 (26/8/2024 - 9/9/2024)

Planning

Se revisaron las historias del *backlog* y se agregaron las de mayor prioridad en el *sprint*.

El equipo repasó y estimó las historias a realizar de acuerdo con el esfuerzo recomendado por la ORT (14 hs semanales por integrante, lo cual equivale a 28 *story points* por iteración, por cada integrante). Esto es una referencia que el equipo tiene en cuenta para el esfuerzo máximo a realizar. Si es necesario un mayor esfuerzo, se puede superar esta cifra.

Los objetivos del *sprint* son: 1) Culminar el *refactor* de la aplicación móvil Flutter. 2) Agregar la vista y lógica (en la aplicación móvil), para poder dejar registro de la cantidad de comensales en cada evento gastronómico del hotel (desayuno, almuerzo, merienda y cena). 3) Agregar vistas y lógica para los módulos de: productos, tipos de productos, proveedores y usuarios. (en la aplicación *web* Angular). 4) Realizar una primera integración entre la *API* de *ChefCheck-Manager*, con el *endpoint* de FacturaLista, para poder obtener las facturas que fueron emitidas hacia el hotel.

Daily 29/8

Francisco comentó sus adelantos. Estuvo trabajando en la gestión de proveedores (*frontend web*).

Andrés completó el *refactor* de la *API .NET*.

Gracias al *refactor* de Andrés, Martín pudo completar su *refactor* de la *app flutter*, y aplicar los cambios que fueron requeridos debido a las modificaciones de la *API*.

Daily 5/9

Martín comentó sus adelantos. Estuvo trabajando en el *backend* de los *endpoints* que gestionan las habitaciones del hotel, el ingreso de comensales al restaurante (para dejar constancia de los mismos), y los check ins de los huéspedes. También Martín trabajó en la aplicación flutter, más precisamente en las pantallas que permiten crear, listar y eliminar los registros de comensales.

Francisco estuvo trabajando en la *web Angular*, en el módulo de gestión de los tipos de productos (categorías).

Andrés cuenta que aguarda por la respuesta de un mail de FacturaLista para poder comenzar con la integración.

Review

El listado de tareas (Ilustración 117) realizadas en este *sprint* y su estado de completitud se puede visualizar en la siguiente tabla (las tareas pendientes al culminar este *sprint* se trasladan automáticamente al *backlog*):

Incidencia	Tarea	User Story	Bug	Estimación en SP	Esfuerzo real en SP	Estado
(1) Automatización de la captura de datos de facturas		X		20	25	En curso
(2) Ingeniería de requerimientos	X			5	-	Pendiente
(3) Refactor Flutter	X			2	8	Completada
(4) Refactor API .NET	X			10	16	Completada
(5) Gestión de proveedores		X		5	9	Completada
(6) Documentación Sprint 5	X			2	2	Completada
(7) Gestión de eventos de Restaurant (backend)		X		4	6	Completada
(8) Gestión de eventos de Restaurant (frontend)		X		3	3	Completada
(9) Arreglar documentación de metodología de trabajo (híbrida)	X			2	-	Pendiente
(10) Gestión de tipos de productos		X		5	2	Completada
(11) Gestión de usuarios		X		3	5	En curso
(12) Gestión/asignación de roles y permisos		X		6	-	Pendiente
(13) Resolve warnings in backend			X	1	-	Pendiente
(14) Gestión de las habitaciones		X		2	2	Completada
(15) Gestión de los check in		X		3	3	Completada
(16) Refactor endpoints PUT	X			1	1	Completada
Total				74	82	10/16 Incidencias completadas

Ilustración 117 - Listado de tareas *Sprint #6*

Burndown Chart

La siguiente gráfica muestra cómo se fueron realizando las entregas de las historias durante el *sprint*:

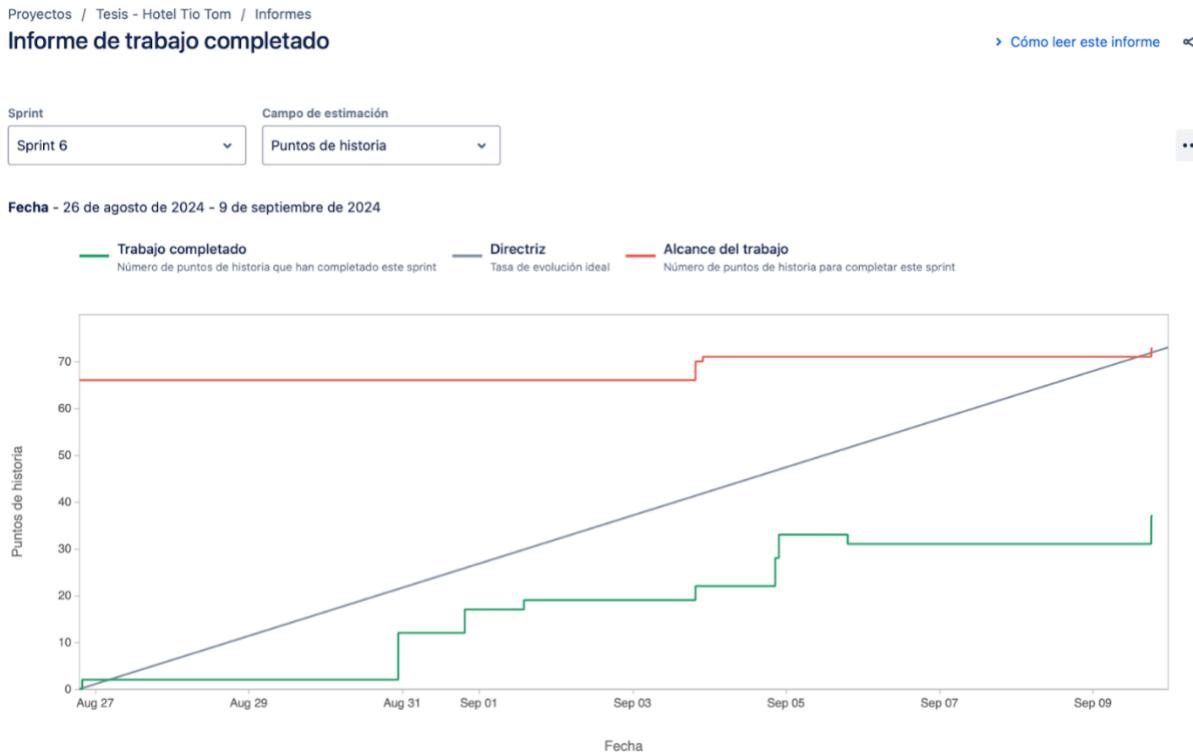


Ilustración 118 - *Burndown chart* del *sprint* #6

El detalle del gráfico (Ilustración 118) dice que el alcance del *sprint* constó de 74 SP, pero se completaron 37 SP, y no 82 como dice en la sección *review*. Eso es porque los 82 son las horas hombre, mientras que los 37 SP corresponden a la estimación de las tareas/*user stories* 100% completadas, sin contar el esfuerzo de tareas/*user stories* sin completar en su totalidad. Por ende, analizando los gráficos sin este contexto, Jira hace aparecer que el equipo no se asoma nunca a los *story points* comprometidos. Si, por ejemplo, una *user story* estimada en 10 *story points*, se encuentra conformada por 2 tareas estimadas en su interior en 5 *story points* cada una, si la segunda tarea no se encuentra terminada al terminar el *sprint*, figura que el equipo no realizó ningún *story point*, pese al haber cumplido con una tarea de las dos que había en dicha *user story*. Por eso el equipo realiza un análisis a mano del esfuerzo de cada *sprint*.

Burnup Chart

La siguiente gráfica muestra cuánto trabajo del total se ha completado:



Ilustración 119 - Burnup chart del sprint #6

El detalle del gráfico (Ilustración 119) dice que quedan 36 SP sin culminar. Estos 36 SP constan de la suma de los SP de estimación de aquellas tareas/*user stories* que no fueron completadas al 100%, osea, aquellas en cualquier otro estado diferente a "Completada". No importa si una *user story* tiene un estado de completitud es de 2 de 3 tareas. El equipo acordó que la misma se completará cuando todo su contenido esté completo.

Velocidad

La diferencia entre los 74 de estimación de horas de trabajo hombre (*story points*), respecto a las horas completadas (82), nos deja una desviación de 8 *story points*.

Es decir, que el sexto *sprint* marca que la velocidad total promedio del equipo es de: (Velocidad *Sprint#1* (45) + Velocidad *Sprint#2* (30) + Velocidad *Sprint#3* (15) + Velocidad *Sprint#4* (70) + Velocidad *Sprint#5* (64) + Velocidad *Sprint#6* (82)) / Cantidad de *Sprints* (6) = 51 *story points*.

El equipo está al tanto de que este número irá variando *sprint* a *sprint*.

Retrospective

El intercambio entre los integrantes del equipo se resumió en el contenido de la siguiente imagen:

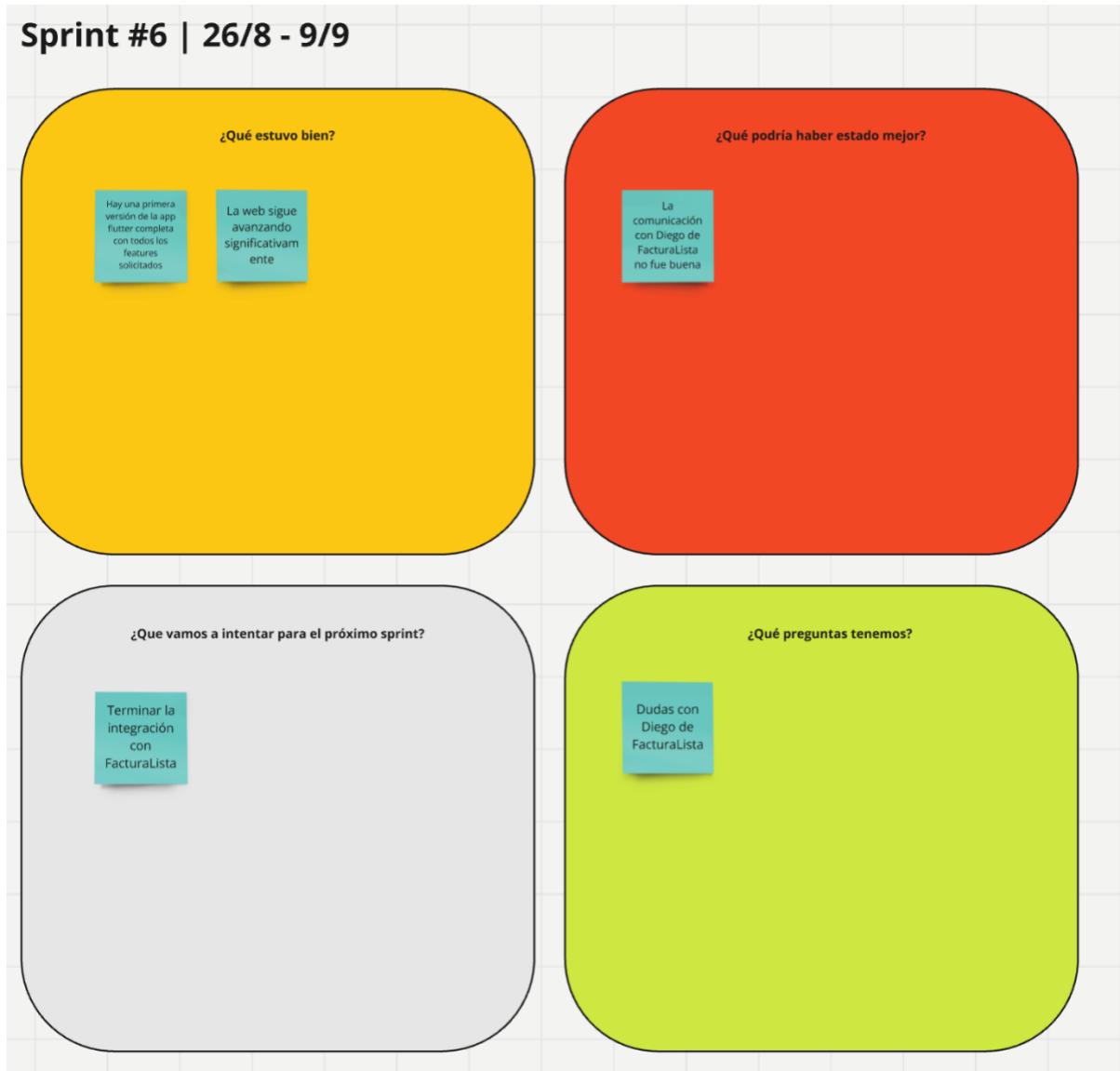


Ilustración 120 - *Sprint retrospective* del *sprint* #6

11.4.11. Documentación de *sprint* #7 (9/9/2024 - 23/9/2024)

Planning

Se revisaron las historias del *backlog* y se agregaron las de mayor prioridad en el *sprint*.

El equipo repasó y estimó las historias a realizar de acuerdo al esfuerzo recomendado por la ORT (14 hs semanales por integrante, lo cual equivale a 28 *story points* por iteración, por cada integrante). Esto es una referencia que el equipo tiene en cuenta para el esfuerzo máximo a realizar. Si es necesario un mayor esfuerzo, se puede superar esta cifra.

Los objetivos del *sprint* son: 1) Eliminar dudas con Diego Dodel de FacturaLista. 2) Realizar una primera integración entre la *API* de *ChefCheck-Manager*, con el *endpoint* de FacturaLista, para poder obtener las facturas que fueron emitidas hacia el hotel. 3) Solucionar *bugs* de la *API*. 4) Agregar notificaciones por email cuando hay bajo stock de un producto. 5) Gestión de los roles en el panel *web* (para evitar que usuarios con cierto rol realicen acciones de otros roles).

Daily 12/9

Martín comentó acerca de sus adelantos: 1) *Refactor* en *endpoints* de tipo *DELETE* de la *API*. 2) Documentación del *sprint* 6. 3) *Testing* de un *bug* que reportó Francisco anteriormente (relacionado a un *endpoint* del registro de proveedores).

Francisco por su lado se encuentra trabajando todavía en la gestión de usuarios.

Andrés fue notificado por parte de FacturaLista, del hecho de que el servicio *REST* para que el sistema *ChefCheck-Manager* pueda consumir, quedará publicado el martes 17 de setiembre.

El equipo se encuentra definiendo si espera al servicio *REST* de esa fecha, o comienza la integración con FacturaLista por medio del servicio *SOAP* existente (ya que hasta el momento la empresa nunca había contado con un servicio *REST*).

Daily 19/9

Martín comentó sus adelantos. Estuvo trabajando en la documentación, en la sección de metodologías ágiles.

Francisco estuvo trabajando en la *web* Angular, en el módulo de gestión de los usuarios.

Andrés cuenta que finalmente pudo comenzar a trabajar en la integración con FacturaLista. Al final la integración será por medio del servicio *SOAP*, y no el servicio *REST*, ya que este último es muy reciente y no muy estable.

Review

El listado de tareas (Ilustración 121) realizadas en este *sprint* y su estado de completitud se puede visualizar en la siguiente tabla (las tareas pendientes al culminar este *sprint* se trasladan automáticamente al *backlog*):

Incidencia	Tarea	User Story	Bug	Estimación en SP	Esfuerzo real en SP	Estado
(1) Gestión de usuarios		X		3	9	Completada
(2) Gestión/asignación de roles y permisos		X		6	-	Pendiente
(3) Automatización de la captura de datos de facturas		X		20	25	En curso
(4) Arreglar documentación de metodología de trabajo (híbrida)	X			2	2	Completada
(5) <i>Resolve warnings in backend</i>	X			1	-	Pendiente
(6) <i>Enabled false when adding new Supplier</i>			X	1	1	Completada
(7) Vista de cantidades y mínimos		X		5	-	Pendiente
(8) Gestión de las habitaciones		X		3	1	En curso
(9) Creación y envío de alertas por cantidades		X		9	9	Completada
(10) Documentación <i>sprint 6</i>	X			1	1	Completada
(11) El RUT son 12 números, por ende una variable <i>int</i> no soporta esa cantidad			X	1	1	Completada
(12) <i>Refactor</i> en <i>responses</i> de <i>endpoints</i> que eliminan registros en la DB	X			1	1	Completada
(13) <i>Refactor</i> de idioma expuesto al usuario en <i>backend</i>			X	0.5	-	Pendiente
(14) El retorno de los <i>endpoints</i> debe ser un objeto			X	0.5	1	Completada
(15) Mensaje en Excepciones dentro de <i>ExceptionFilter</i>			X	1	1	Completada
(16) Cuando se elimina un movimiento de			X	1	-	Pendiente

inventario, no se restaura el stock						
(17) GuestEntry y Room deben tener borrado lógico, con Enabled	X			1	3	Completada
(18) Separar en GetAllEnabled y GetAll común	X			2	-	Pendiente
Total				59	55	10/18 Incidencias completadas

Ilustración 121 - Listado de tareas *Sprint #7*

Burndown Chart

La siguiente gráfica muestra cómo se fueron realizando las entregas de las historias durante el *sprint*:



Ilustración 122 - *Burndown chart* del *sprint #7*

El detalle del gráfico (Ilustración 122) dice que el alcance del *sprint* constó de 52 SP, pero en realidad fue de 59 SP el alcance. Esto sucede, debido a que la tarea "Separar en GetAllEnabled y GetAll común", el equipo olvidó indicar los SP de estimación para dos tareas. Por eso en realidad son 59 SP.

Según Jira, el equipo únicamente completó 20.5 SP, y no 55 como dice en la sección *review*. Eso se debe a que 55 son las horas hombre, mientras que los 20.5 SP corresponden a la estimación de las tareas/*user stories* 100% completadas, sin contar el esfuerzo de tareas/*user stories* sin completar en su totalidad. Por ende, analizando los gráficos sin este contexto, Jira hace aparentar que el equipo no se asoma jamás a los *story points* comprometidos. Si por ejemplo, una *user story* estimada en 10 *story points*, se encuentra conformada por 2 tareas estimadas en su interior en 5 *story points* cada una, si la segunda tarea no se encuentra terminada al terminar el *sprint*, figura que el equipo no realizó ningún *story point*, pese al haber cumplido con una tarea de las dos que había en dicha *user story*. Por eso el equipo realiza un análisis a mano del esfuerzo de cada *sprint*.

Burnup Chart

La siguiente gráfica muestra cuánto trabajo del total se ha completado:



Ilustración 123 - *Burnup chart* del *sprint* #7

El detalle del gráfico (Ilustración 123) dice que quedan 31.5 SP sin culminar. Estos 31.5 SP constan de la suma de los SP de estimación de aquellas tareas/*user stories* que no fueron completadas al 100%, osea, aquellas en cualquier otro estado diferente a "Completada". No importa si una *user story* tiene un estado de completitud es de 2 de 3 tareas. El equipo acordó que la misma se completará cuando todo su contenido esté completo.

Velocidad

La diferencia entre los 59 de estimación de horas de trabajo hombre (*story points*), respecto a las horas completadas (55), nos deja una desviación de 4 *story points*.

Es decir, que el séptimo *sprint* marca que la velocidad total promedio del equipo es de: (Velocidad *Sprint#1* (45) + Velocidad *Sprint#2* (30) + Velocidad *Sprint#3* (15) + Velocidad *Sprint#4* (70) + Velocidad *Sprint#5* (64) + Velocidad *Sprint#6* (82) + Velocidad *Sprint#7* (55)) / Cantidad de *Sprints* (7) = 51.5 *story points*.

El equipo está al tanto de que este número irá variando *sprint* a *sprint*.

Retrospective

El intercambio entre los integrantes del equipo se resumió en el contenido de la siguiente imagen:

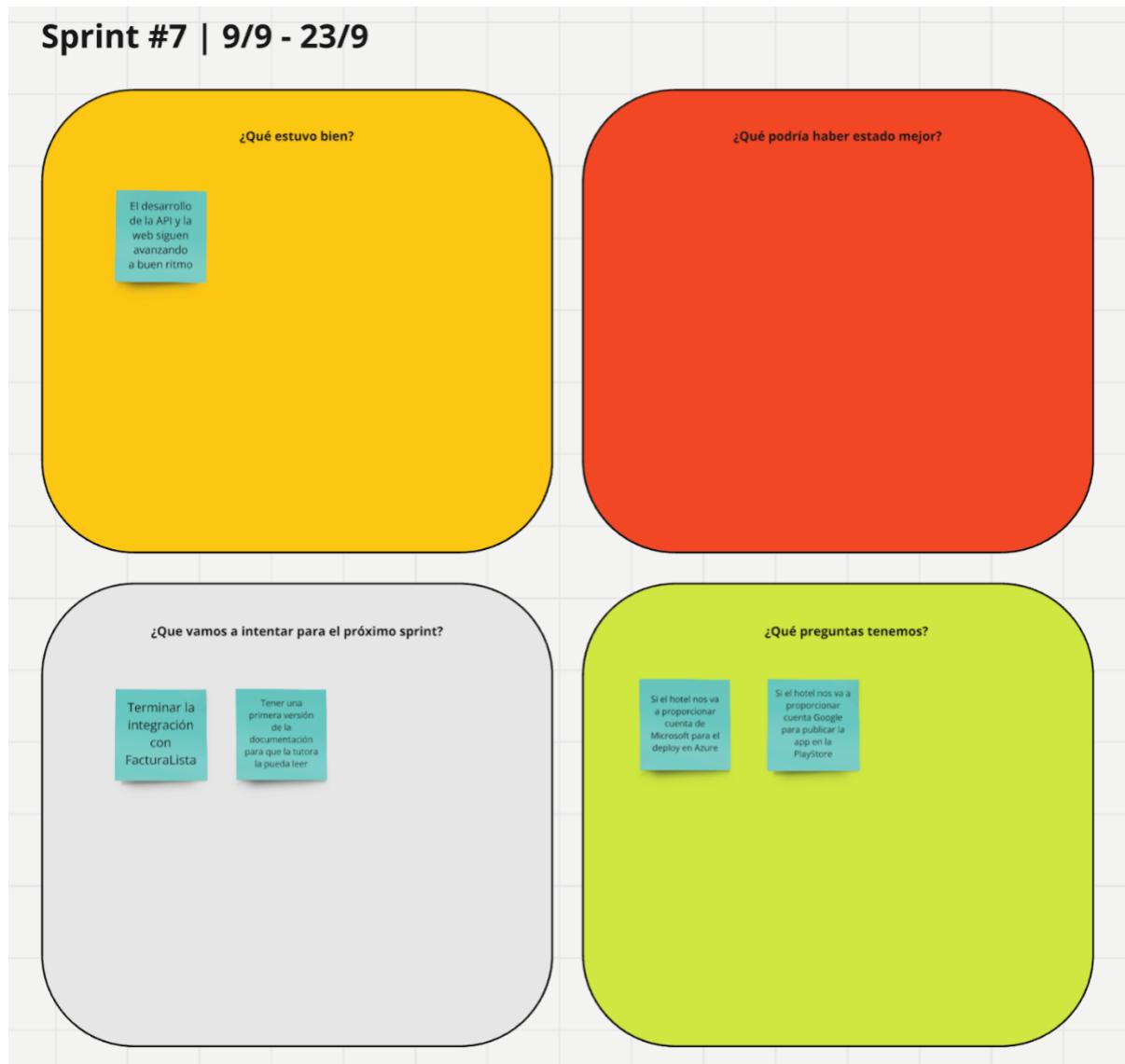


Ilustración 124 - *Sprint retrospective* del *sprint #7*

11.4.12. Documentación de *sprint* #8 (23/9/2024 - 7/10/2024)

Planning

Se revisaron las historias del *backlog* y se agregaron las de mayor prioridad en el *sprint*.

El equipo repasó y estimó las historias a realizar de acuerdo al esfuerzo recomendado por la ORT (14 hs semanales por integrante, lo cual equivale a 28 *story points* por iteración, por cada integrante). Esto es una referencia que el equipo tiene en cuenta para el esfuerzo máximo a realizar. Si es necesario un mayor esfuerzo, se puede superar esta cifra.

Los objetivos del *sprint* son: 1) Seguir avanzando en la integración con FacturaLista. 2) Avanzar con la documentación pensando en la entrega administrativa de gestión. 3) Arreglar *bugs*. 4) Pequeñas refactorizaciones en la *API*.

Daily 26/9

Martín comenta que está trabajando en la documentación, sobre todo en la sección que estaba menos desarrollada al momento (arquitectura).

Francisco estuvo trabajando tanto en *refactor* de la *API* como desarrollando el feature de gestión de las habitaciones en la aplicación *web* Angular.

Andrés se encuentra integrando la *API* del sistema de equipo, con el servicio *SOAP* de FacturaLista.

Segunda Revisión 30/9

La segunda revisión, por motivos ajenos al equipo, se vio forzada a ser suspendida, sin nueva fecha agendada por el momento.

Daily 3/10

Martín nos comenta que sigue dedicándose exclusivamente a la documentación, refinando diagramas que anteriormente fueron hechos. También trabajó revisando la sección de objetivos en la documentación.

Francisco siguió trabajando en la *web* Angular, en lo que corresponde a la gestión de productos.

Andrés comenzó a documentar junto a Martín. Comenzó revisando las secciones que él previamente había realizado. Arregló con Francisco para que sea él quien continúe la integración con FacturaLista.

Review

El listado de tareas (Ilustración 125) realizadas en este *sprint* y su estado de completitud se puede visualizar en la siguiente tabla (las tareas pendientes al culminar este *sprint* se trasladan automáticamente al *backlog*):

Incidencia	Tarea	User Story y	Bug	Estimación en SP	Esfuerzo real en SP	Estado
(1) Refactor de idioma expuesto al usuario en <i>backend</i>			X	0.5	6	Completada
(2) Cuando se elimina un movimiento de inventario, no se restaura el <i>stock</i>			X	1	1	Completada
(3) Automatización de la captura de datos de facturas		X		20	31	En curso
(4) Gestión de las habitaciones		X		3	1	Completada
(5) Gestión/asignación de roles y permisos		X		6	-	Pendiente
(6) Separar en GetAllEnabled y GetAll comun	X			2	2	Completada
(7) Resolve warnings in backend			X	1	-	Pendiente
(8) Documentación del <i>sprint 7</i>	X			1	1	Completada
(9) Deployar DB	X			2	-	En espera
(10) Deployar API	X			2	-	En espera
(11) Deployar Web	X			2	-	En espera
(12) Deployar app	X			2	-	En espera
(13) Documentar paso a paso de todos los <i>deploys</i>	X			5	5	Completada
(14) Presentación segunda revisión	X			5	5	Completada
(15) Ingeniería de requerimientos	X			5	0.5	En curso
(16) Visualización de reportes (<i>frontend</i>)		X		8	-	Pendiente
(17) Visualización de Reportes (<i>backend</i>)		X		15	-	Pendiente
(18) Documentar integración con Factura Lista	X			2	2	Completada
(19) Actualizar UML's que ya estaban hechos	X			5	5	Completada

(20) Actualizar doc de arquitectura	X			7	7	Completada
(21) Diagrama MER base de datos	X			2	2	Completada
(22) Documentar y describir MER en la documentación	X			1	1	Completada
(23) Analizar cumplimiento de los objetivos	X			3	3	Completada
(24) Nuevos diagramas de secuencia de casos de uso clave	X			2	2	Completada
(25) Bug cuando se crea un InventoryMovement			X	1	3	Completada
(26) Bug en modelo de Room			X	1	1	Completada
(27) Escribir bien la palabra "Comensal" en la app	X			1	1	Completada
(28) Bug login en app			X	1	1	Completada
(29) Gestión de productos		X		7	5	Completada
(30) Refactor asynchronous EmailService and LoggerService	X			4	2	En curso
(31) Implementar el uso de paginación	X			2	-	Pendiente
(32) Análisis de heurísticas de Nielsen	X			2	2	Completada
(33) lowStockUsers not taking into account in product.type.component			X	5	2	En curso
(34) Refactor to <mat-form-field> in Angular	X			4	-	Pendiente
Total				130.5	91.5	20/34 Incidencias completadas

Ilustración 125 - Listado de tareas *Sprint #8*

Burndown Chart

La siguiente gráfica muestra cómo se fueron realizando las entregas de las historias durante el *sprint*:

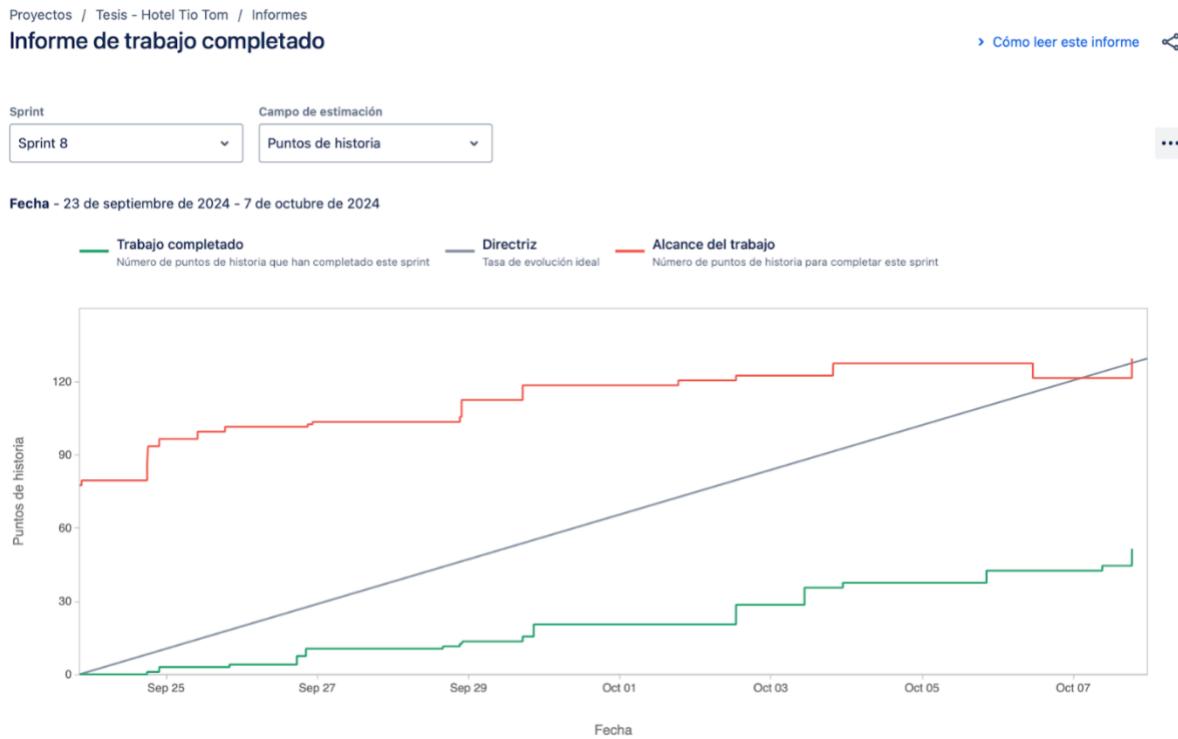


Ilustración 126 - *Burndown chart* del *sprint* #8

El detalle del gráfico (Ilustración 126) dice que el alcance del *sprint* constó de 129.5 SP, pero en realidad fue de 130.5 SP el alcance. Esto sucede, debido a que, al momento de crear el *sprint* en Jira, no se asignó los *story points* estimados de una tarea. El equipo se olvidó de setearlo en el momento, por eso el alcance real es de 130.5 SP.

Según Jira, el equipo únicamente completó 51.5 SP, y no 91.5 como dice en la sección *review*. Eso se debe a que 91.5 son las horas hombre, mientras que los 51.5 SP corresponden a la estimación de las tareas/*user stories* 100% completadas, sin contar el esfuerzo de tareas/*user stories* sin completar en su totalidad. Por ende, analizando los gráficos sin este contexto, Jira hace aparecer que el equipo no se asoma jamás a los *story points* comprometidos. Si, por ejemplo, una *user story* estimada en 10 *story points*, se encuentra conformada por 2 tareas estimadas en su interior en 5 *story points* cada una, si la segunda tarea no se encuentra terminada al terminar el *sprint*, figura que el equipo no realizó ningún *story point*, pese al haber cumplido con una tarea de las dos que había en dicha *user story*. Por eso el equipo realiza un análisis a mano del esfuerzo de cada *sprint*.

Burnup Chart

La siguiente gráfica muestra cuánto trabajo del total se ha completado:



Ilustración 127 - Burnup chart del sprint #8

El detalle del gráfico (Ilustración 127) dice que quedan 78 SP sin culminar. Estos 78 SP constan de la suma de los SP de estimación de aquellas tareas/*user stories* que no fueron completadas al 100%, osea, aquellas en cualquier otro estado diferente a "Completada". No importa si una *user story* tiene un estado de completitud es de 2 de 3 tareas. El equipo acordó que la misma se completará cuando todo su contenido esté completo.

Velocidad

La diferencia entre los 130.5 de estimación de horas de trabajo hombre (*story points*), respecto a las horas completadas (91.5), nos deja una desviación de 39 *story points*.

Es decir, que el octavo *sprint* marca que la velocidad total promedio del equipo es de: (Velocidad *Sprint#1* (45) + Velocidad *Sprint#2* (30) + Velocidad *Sprint#3* (15) + Velocidad *Sprint#4* (70) + Velocidad *Sprint#5* (64) + Velocidad *Sprint#6* (82) + Velocidad *Sprint#7* (55) + Velocidad *Sprint#8* (91.5)) / Cantidad de *Sprints* (8) = 56.5 *story points*.

El equipo está al tanto de que este número irá variando *sprint* a *sprint*.

Retrospective

El intercambio entre los integrantes del equipo se resumió en el contenido de la siguiente imagen:

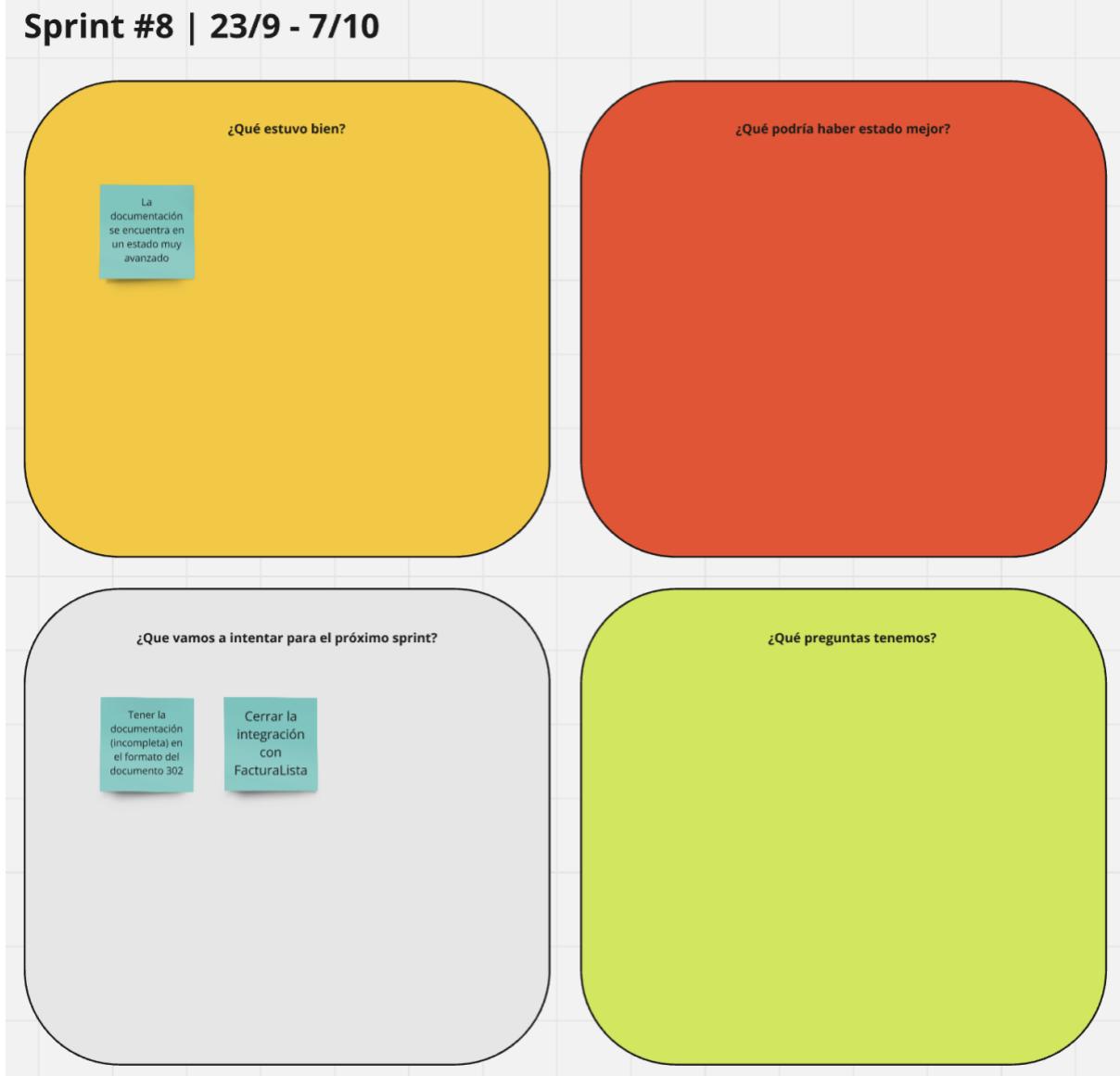


Ilustración 128 - *Sprint retrospective* del *sprint #8*

11.4.13. Documentación de *sprint* #9 (7/10/2024 - 17/10/2024)

Planning

Se revisaron las historias del *backlog* y se agregaron las de mayor prioridad en el *sprint*.

El equipo repasó y estimó las historias a realizar de acuerdo al esfuerzo recomendado por la ORT (14 hs semanales por integrante, lo cual equivale a 28 *story points* por iteración, por cada integrante). Esto es una referencia que el equipo tiene en cuenta para el esfuerzo máximo a realizar. Si es necesario un mayor esfuerzo, se puede superar esta cifra.

Los objetivos del *sprint* son: 1) Seguir avanzando en la integración con FacturaLista. 2) Avanzar con la documentación pensando en la entrega administrativa de gestión. 3) Arreglar *bugs*. 4) Pequeñas refactorizaciones en la *API*.

Daily 10/10

Martín y Andrés comentan que sigue trabajando en conjunto en lo que corresponde con la documentación, ya sea agregando nuevos contenidos, o modificando el documento para cumplir con el formato adecuado.

Francisco nos cuenta que tiene adelantos de la funcionalidad de integración con FacturaLista para la obtención de las facturas. Se encuentra a punto de terminar dicha funcionalidad.

Daily 15/10

Martín y Andrés comentan que sigue trabajando en conjunto en lo que corresponde con la documentación, ya sea agregando nuevos contenidos, o modificando el documento para cumplir con el formato adecuado.

Francisco terminó la integración con FacturaLista y sigue desarrollando la parte de la *app web* de reportes y vista de facturas. Luego seguiría con la documentación de ingeniería de requerimientos para que quede pronta en el documento.

Review

El listado de tareas (Ilustración 129) realizadas en este *sprint* y su estado de completitud se puede visualizar en la siguiente tabla (las tareas pendientes al culminar este *sprint* se trasladan automáticamente al *backlog*):

Incidencia	Tarea	User Story	Bug	Estimación en SP	Esfuerzo real en SP	Estado
(1) Problema y justificación de la solución	X			3	3	En revisión
(2) Definición de desafíos	X			4	4	Completada
(3) Visualización de Reportes (<i>backend</i>)		X		15	-	Pendiente
(4) <i>Resolve warnings in backend</i>			X	1	-	Pendiente
(5) <i>Refactor to <mat-form-field> in Angular</i>	X			4	-	Pendiente
(6) lowStockUsers not taking into account in product.type.component			X	5	2	En curso
(7) <i>Refactor asynchronous EmailService and LoggerService</i>	X			4	4	Pendiente
(8) Visualización de reportes (<i>frontend</i>)		X		8	-	Pendiente
(9) Automatización de la captura de datos de facturas		X		20	39	En curso
(10) Ingeniería de requerimientos (documentación)	X			5	0.5	En curso
(11) Gestión/asignación de roles y permisos		X		6	-	Pendiente
(12) Historial de compras		X		3	6	En curso
(13) Gestión de riesgo (análisis)	X			4	-	Pendiente
(14) Gestión de compras		X		4	4	Completada
(15) Atributos de calidad en base a requerimientos no funcionales	X			4	4	Completada
(16) Formateo del doc final (302)	X			25	25	Completada
(17) Documentar calidad				8	8	Completada
(18) Gestión de la configuración	X			3	3	Completada

(19) Métricas de calidad	X			3	-	Pendiente
(20) Tiempo de lectura y escritura en calidad	X			1	-	Pendiente
(21) Actualizar doc de marco metodológico	X			3	3	Completada
(22) Hacer links para los anexos cada vez que se mencionan	X			2	2	Completada
(23) Actualizar doc de gestión de proyecto	X			3	3	Completada
(24) Documentación del <i>sprint 8</i>	X			1	1	Completada
(25) Lecciones aprendidas	X			2	2	Completada
(26) Documentar las referencias bibliográficas previamente utilizadas	X			2	2	Completada
(27) Palabras clave	X			1	1	Completada
(28) Agradecimientos	X			1	1	Completada
(29) <i>Abstract</i>	X			1	1	Completada
(30) Imágenes actualizadas del resto de los diagramas	X			3	3	Completada
(31) Imágenes de diagramas de secuencia	X			2	2	Pendiente
(32) Poner en itálica las palabras en inglés	X			2	2	Completada
(33) Enumerar las imágenes de todo el documento	X			2	2	Completada
(34) GetProducts endpoint bug en app móvil			X	1	1	Completada
Total				156	128.5	19/34 Incidencias completadas

Ilustración 129 - Listado de tareas *Sprint #9*

Burndown Chart

La siguiente gráfica muestra cómo se fueron realizando las entregas de las historias durante el *sprint*:

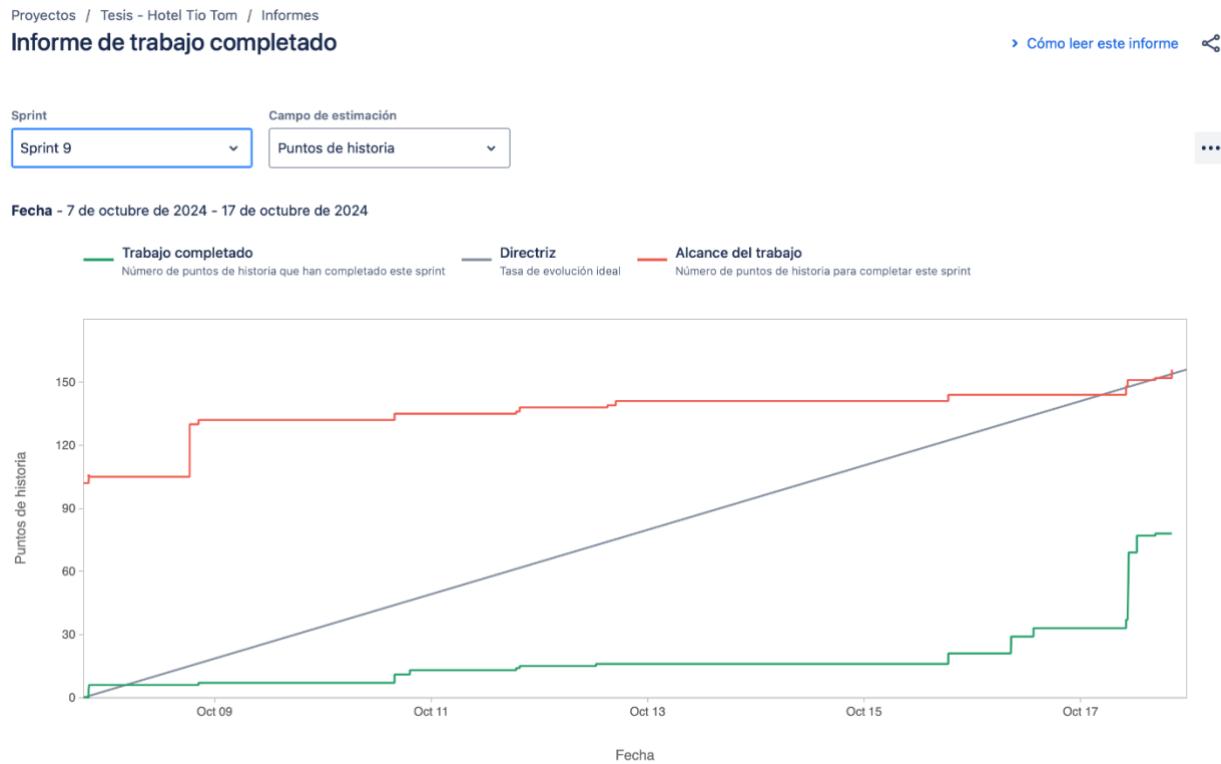


Ilustración 130 - *Burndown chart* del *sprint* #9

El detalle del gráfico (Ilustración 130) dice que el alcance del *sprint* constó de 156 SP, pero se completaron 78 SP, y no 128.5 como dice en la sección *review*. Eso es porque los 128.5 son las horas hombre, mientras que los 78 SP corresponden a la estimación de las tareas/*user stories* 100% completadas, sin contar el esfuerzo de tareas/*user stories* sin completar en su totalidad. Por ende, analizando los gráficos sin este contexto, Jira hace aparecer que el equipo no se asoma nunca a los *story points* comprometidos. Si, por ejemplo, una *user story* estimada en 10 *story points*, se encuentra conformada por 2 tareas estimadas en su interior en 5 *story points* cada una, si la segunda tarea no se encuentra terminada al terminar el *sprint*, figura que el equipo no realizó ningún *story point*, pese al haber cumplido con una tarea de las dos que había en dicha *user story*. Por eso el equipo realiza un análisis a mano del esfuerzo de cada *sprint*.

Burnup Chart

La siguiente gráfica muestra cuánto trabajo del total se ha completado:



Ilustración 131 - *Burnup chart* del *sprint #9*

El detalle del gráfico (Ilustración 131) dice que quedan 78 SP sin culminar. Estos 78 SP constan de la suma de los SP de estimación de aquellas tareas/*user stories* que no fueron completadas al 100%, osea, aquellas en cualquier otro estado diferente a "Completada". No importa si una *user story* tiene un estado de completitud es de 2 de 3 tareas. El equipo acordó que la misma se completará cuando todo su contenido esté completo.

Velocidad

La diferencia entre los 156 de estimación de horas de trabajo hombre (*story points*), respecto a las horas completadas (128.5), nos deja una desviación de 27.5 *story points*.

Es decir, que el noveno *sprint* marca que la velocidad total promedio del equipo es de: (Velocidad *Sprint#1* (45) + Velocidad *Sprint#2* (30) + Velocidad *Sprint#3* (15) + Velocidad *Sprint#4* (70) + Velocidad *Sprint#5* (64) + Velocidad *Sprint#6* (82) + Velocidad *Sprint#7* (55) + Velocidad *Sprint#8* (91.5) + Velocidad *Sprint#9* (128.5)) / Cantidad de *Sprints* (9) = 64.5 *story points*.

El equipo está al tanto de que este número irá variando *sprint* a *sprint*.

Retrospective

El intercambio entre los integrantes del equipo se resumió en el contenido de la siguiente imagen:

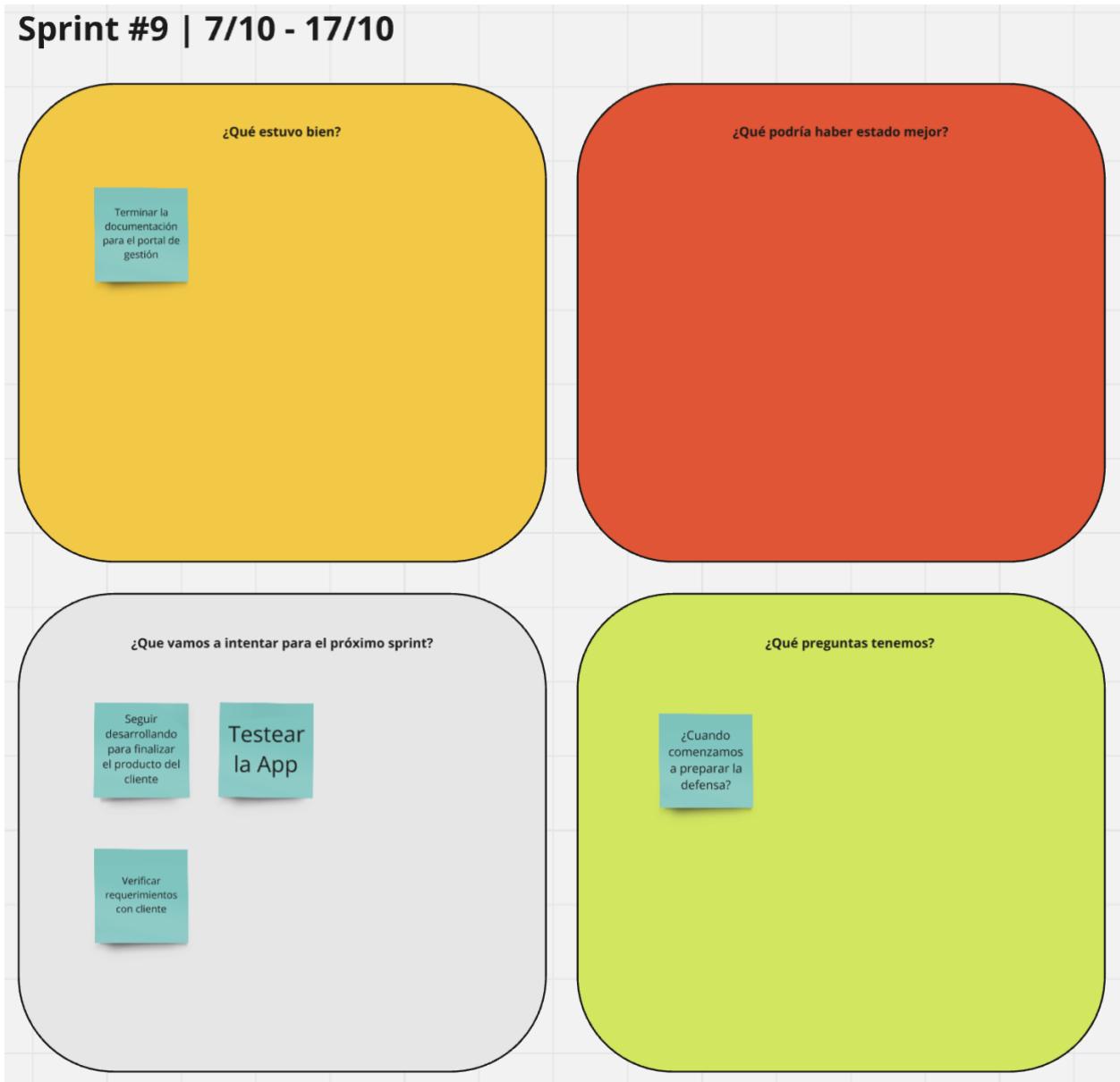


Ilustración 132 - Sprint retrospective del sprint #9

11.5. Plan de calidad

Actividad	Técnica	Resultado	Recursos	Responsable	Participantes
Empatía					
Relevamiento de necesidades y restricciones del cliente	Reuniones con el cliente	Documento de necesidades y restricciones del cliente	-	Equipo	Equipo, cliente
Relevamiento de necesidades del usuario final	Entrevistas con potenciales usuarios	Documento de necesidades del usuario	-	Equipo	Equipo, cliente
Ingeniería de requerimientos					
Especificación de requerimientos	Board compartido entre integrantes de equipo, historias de usuario	Documento de especificación de requerimientos	Necesidades de cliente y usuario, análisis de productos similares, User Journey	Equipo	Equipo, cliente
Revisión de requerimientos	Revisión entre equipo y cliente	Requerimientos revisados	Documento de especificación de requerimientos	Equipo	Equipo, cliente
Validación de requerimientos con el cliente	Reunión entre equipo y el cliente	Documento final con requerimientos con validación del cliente	Documento de especificación de requerimientos (Ya revisados)	Equipo	Equipo, cliente
Validación de requerimientos con usuarios	Reuniones y entrevistas con futuros usuarios	Documento de validación de requerimientos de usuario	Documento de especificación de necesidad	Equipo	Equipo, futuros usuarios

			des de usuario		
Priorización de requerimientos	Reunión con el cliente	Ítems de product backlog con prioridad asignada	Documento de especificación de requerimientos	Equipo	Equipo, cliente
Validación de prototipos de diseño	Reunión con cliente	Prototipos de diseño validados	Prototipos de diseño	Equipo	Equipo, cliente
Diseño y arquitectura					
Estudio de tecnologías para <i>backend</i>	Investigación de oferta tecnológica y composición de informe comparativo	Documento de análisis y comparación de tecnologías <i>backend</i>	Información disponible de tecnologías, especificación de requerimientos	Equipo	Equipo
Estudio de tecnologías para <i>frontend</i>	Investigación de oferta tecnológica y composición de informe comparativo	Documento de análisis y comparación de tecnologías <i>frontend</i>	Información disponible de tecnologías, especificación de requerimientos	Equipo	Equipo
Definición de estándares de codificación	Investigación de las mejores prácticas según tecnología	Documento con estándares de codificación	Documento de análisis y comparación de tecnologías	Equipo	Equipo
Diseño de la arquitectura	Trabajo cooperativo del equipo de desarrollo	Documento de arquitectura	Especificación de requerimientos, informe de tecnologías	Equipo	Equipo

Desarrollo					
Capacitación del equipo de desarrollo	Estudio necesario	Equipo más capacitado	Informe de tecnologías, documentación de estas, material público	Equipo	Equipo
Codificación de los requerimientos representados en las historias de usuario	Codificar	Distintos proyectos con su código fuente	Documento de estándares de codificación, documento de especificación de requerimientos	Equipo	Equipo
Revisiones de código de cada historia de usuario	Revisión entre pares (Miembros del equipo de desarrollo)	Distintos proyectos con su código fuente revisado	Código fuente, documento de estándares de codificación, documento de especificación de requerimientos	Equipo	Equipo
Pruebas					
Diseño de casos de prueba funcionales	Reunión de integrantes de equipo de desarrollo	Documento de especificación de casos de prueba funcionales	Documento de especificación de requerimientos	Equipo	Equipo
Ejecución de casos de prueba funcionales	Ejecución manual de casos de prueba (Ambiente adecuado)	Documento con resultados de casos de prueba funcionales	Documento de especificación de	Equipo	Equipo

			casos de prueba funcionales		
Diseño y codificación de pruebas unitarias	Codificar	Código fuente de pruebas unitarias	Documento de especificación de requerimientos	Equipo	Equipo
Ejecución de pruebas unitarias	Correr herramienta de pruebas	Documento con resultados de pruebas unitarias	Documento de especificación de requerimientos	Equipo	Equipo
Diseño y codificación de pruebas de integración	Codificar	Código fuente de pruebas de integración	Documento de especificación de requerimientos	Equipo	Equipo
Ejecución de pruebas de integración	Correr herramienta de pruebas	Documento con resultados de pruebas de integración	Documento de especificación de requerimientos	Equipo	Equipo
Diseño y codificación de pruebas de carga	Codificar	Ejecutables de pruebas de carga	Documento de especificación de requerimientos (Performance)	Equipo	Equipo
Ejecución de pruebas de carga	Ejecución manual de pruebas de carga (Ambiente adecuado)	Documento con resultados de pruebas de carga	Ejecutables de pruebas de carga	Equipo	Equipo
Diseño de pruebas de usabilidad	Reunión de integrantes de equipo de desarrollo	Documento de especificación de escenarios de prueba de usabilidad.	Documento de especificación de requerimientos (Usabilidad)	Equipo	Equipo

Ejecución de pruebas de usabilidad con usuarios	Pruebas con usuarios	Documento con resultados de pruebas de usabilidad	Documento de especificación de escenarios de prueba de usabilidad.	Equipo	Equipo, cliente, usuarios
Ejecución de pruebas de portabilidad	<i>Testing</i> exploratorio (<i>web</i> y móvil) en distintos dispositivos	Documento con resultados de las pruebas de portabilidad	Documento de especificación de requerimientos (Portabilidad)	Equipo	Equipo
Actividades de apoyo					
<i>Sprint planning</i>	Reunión de integrantes de equipo de desarrollo	Historias de usuario para el <i>sprint</i> a planificar	Documento de especificación de requerimientos, product backlog	Martín	Equipo
<i>Sprint retrospective</i>	Reunión de integrantes de equipo de desarrollo	Potenciales cambios en interacciones, artefactos, procesos, historias de usuario, forma de trabajo	Ánálisis de los procesos, interacciones, herramientas y definición of done	Martín	Equipo
<i>Sprint review</i>	Reunión de integrantes de equipo de desarrollo	Product backlog actualizado y con prioridades corregidas	Historias de usuario para el <i>sprint</i> a planificar	Martín	Equipo

Análisis de riesgos	Reunión de integrantes de equipo de desarrollo	Documento con el plan de riesgos del proyecto	Documento de especificación de requerimientos, informe de tecnologías, análisis de factores externos	Fransisco	Equipo
Análisis de SQA	Reunión de integrantes de equipo de desarrollo	Documento con plan de gestión de SQA	Documento de especificación de necesidades del cliente, informe de tecnologías, documento de especificación de requerimientos	Andrés	Equipo

Ilustración 133 - Plan de calidad del equipo

11.6. Atributos de calidad

RNF-1 – Seguridad

2. **Atributo de calidad:** Seguridad
 - c. **Definición:** Garantizar la protección de los datos sensibles en tránsito y en reposo, protegiendo la confidencialidad, integridad y autenticidad de la información.
 - d. **Recomendaciones para cumplirlo:**
 - iv. Implementar HTTPS: Configurar un certificado SSL para todas las comunicaciones entre el servidor y los clientes, garantizando que todos los datos transmitidos estén cifrados.
 - v. Utilizar JWT para autenticación: Implementar JSON Web Tokens (JWT) para la autenticación de usuarios. Asegurarse de usar algoritmos seguros como RSA o HS256.
 - vi. Cifrado de datos en reposo: Implementar cifrado AES-256 o superior para todos los datos sensibles almacenados en bases de datos y archivos. Verificar que las claves de cifrado estén protegidas adecuadamente.
 - vii. Configurar expiración para tokens JWT: Establecer una expiración clara para los tokens JWT para reducir riesgos de reutilización de tokens comprometidos.

RNF-2 – Disponibilidad

3. **Atributo de calidad:** Disponibilidad
 - e. **Definición:** Garantizar que el sistema esté disponible para los usuarios al menos el 99% del tiempo durante la temporada alta.
 - f. **Recomendaciones para cumplirlo:**
 - viii. Configurar monitoreo de disponibilidad: Implementar herramientas como UptimeRobot o Pingdom para monitorear la disponibilidad del sistema en tiempo real.
 - ix. Planificar tiempos de mantenimiento: Definir ventanas de mantenimiento programadas y asegurar que los tiempos de inactividad no afecten la disponibilidad crítica.
 - x. Implementar redundancia: Utilizar servidores redundantes o平衡adores de carga para asegurar que, en caso de fallos, el sistema siga operativo.

RNF-3 – Escalabilidad

4. **Atributo de calidad:** Escalabilidad
 - g. **Definición:** Asegurar que el sistema pueda escalar horizontalmente para manejar aumentos en la carga de trabajo sin afectar el rendimiento.

h. Recomendaciones para cumplirlo:

- xii. Configurar escalado automático: Implementar soluciones de autoescalado en la infraestructura (AWS Auto Scaling, Azure Scale Sets) para manejar picos de tráfico.
- xiii. Realizar pruebas de carga: Usar herramientas como Apache JMeter o Locust para realizar simulaciones de carga y verificar que el sistema maneja el aumento de usuarios y transacciones.

RNF-4 – Rendimiento

5. Atributo de calidad: Rendimiento

- i. **Definición:** Garantizar que el tiempo de respuesta para transacciones y consultas sea rápido y adecuado para una buena experiencia de usuario. Se establece 3.0 segundos como tiempo objetivo máximo para las operaciones de lectura y de 2.5 segundos para las operaciones de escritura.

j. Recomendaciones para cumplirlo:

- xiii. Optimizar el código: Revisar y optimizar el código para reducir la latencia en las operaciones de ingreso, modificación y consulta de datos.
- xiv. Configurar pruebas de estrés: Ejecutar pruebas de rendimiento utilizando herramientas como Apache JMeter para asegurarse de que el tiempo de respuesta se mantenga por debajo de los tiempos establecidos.

RNF-5 – Usabilidad

6. Atributo de calidad: Usabilidad

- k. **Definición:** Asegurar que la interfaz de usuario sea intuitiva, accesible y brinde una experiencia satisfactoria para usuarios con diferentes niveles de habilidad tecnológica.

l. Recomendaciones para cumplirlo:

- xv. Aplicar heurísticas de Nielsen: Seguir los principios de usabilidad de Nielsen en el diseño de la interfaz, asegurando que sea fácil de aprender, eficiente de usar y que proporcione retroalimentación adecuada.
- xvi. Realizar pruebas de usuario: Llevar a cabo pruebas de usabilidad con usuarios reales para obtener *feedback* sobre la facilidad de uso del sistema.

RNF-6 – Compatibilidad

7. Atributo de calidad: Compatibilidad

- m. **Definición:** Asegurar que el sistema funcione correctamente en múltiples dispositivos y navegadores.

n. Recomendaciones para cumplirlo:

- xvii. Realizar pruebas de navegador: Probar compatibilidad en los navegadores más populares como lo son Google Chrome, Firefox y Safari.
- xviii. Probar en dispositivos reales: Ejecutar la aplicación en dispositivos Android (tabletas) y validar que funciona correctamente en modo horizontal.

RNF-7 – Extensibilidad

8. Atributo de calidad: Extensibilidad

- o. **Definición:** Facilitar la integración de nuevos módulos o funcionalidades en el futuro sin afectar el funcionamiento del sistema actual.
- p. **Recomendaciones para cumplirlo:**
 - xix. Diseñar arquitectura modular: Asegurarse de que el sistema esté diseñado con una arquitectura modular, lo que facilita la adición de nuevos componentes sin afectar los ya existentes.
 - xx. Documentar el código: Mantener una documentación clara que permita a otros desarrolladores integrar nuevas funcionalidades de manera eficiente.
 - xxi. Planificar la integración de *machine learning*: Proporcionar interfaces y puntos de integración para los futuros algoritmos de *machine learning*, de manera que puedan ser añadidos sin modificaciones importantes.

RNF-8 – Mantenibilidad

9. Atributo de calidad: Mantenibilidad

- q. **Definición:** Facilitar el mantenimiento y la actualización del sistema a través de una arquitectura clara y bien documentada.
- r. **Recomendaciones para cumplirlo:**
 - xxii. Seguir estándares de codificación: Implementar buenas prácticas y estándares de codificación (ej. SOLID, DRY) que faciliten la comprensión y modificación del código.
 - xxiii. Mantener una arquitectura modular: Asegurar que el sistema esté dividido en módulos independientes, lo que facilita el mantenimiento y la actualización.
 - xxiv. Planificar ventanas de actualización: Definir momentos específicos para realizar actualizaciones sin afectar la disponibilidad del sistema durante el horario de trabajo del hotel.

RNF-9 – Recuperación ante fallos

10. Atributo de calidad: Tolerancia a fallos / Recuperación

s. **Definición:** Garantizar que el sistema pueda recuperarse rápidamente en caso de incidentes críticos.

t. **Recomendaciones para cumplirlo:**

xxv. Implementar copias de seguridad regulares: Configurar respaldos automáticos de la base de datos y los archivos críticos con regularidad.

RNF-10 – Integración

11. Atributo de calidad: Interoperabilidad

u. **Definición:** Asegurar que el sistema se integre correctamente con servicios externos, como FacturaLista.

v. **Recomendaciones para cumplirlo:**

xxvi. Diseñar interfaces de integración: Implementar APIs claras que permitan la integración con otros sistemas, asegurando que los datos se transfieran correctamente.

xxvii. Probar la interoperabilidad: Ejecutar pruebas con los servicios externos para garantizar que la integración funcione sin problemas.

RNF-11 – Monitorización y registro

12. Atributo de calidad: Monitorización

w. **Definición:** Asegurar que el sistema registre y monitoree errores de manera efectiva, facilitando la detección y resolución de problemas.

x. **Recomendaciones para cumplirlo:**

xxviii. Configurar alertas automáticas: Configurar alertas en tiempo real para que el equipo de desarrollo sea notificado inmediatamente cuando ocurra un error crítico.

xxix. Almacenar los logs de manera segura: Asegurarse de que los registros se almacenan de forma segura en una base de datos y que solo personal autorizado tenga acceso a ellos.

11.7. Conclusiones

11.7.1. Objetivo "Disfrutar y aprender durante el desarrollo del proyecto"

¿Cómo calificarías el nivel de apoyo y ayuda que recibiste de los otros miembros del equipo?



Ilustración 134 - Pregunta 1 formulario de equipo

¿Qué tan clara fue la asignación de roles y responsabilidades dentro del equipo?



Ilustración 135 - Pregunta 2 formulario de equipo

¿En qué medida sentiste que tus ideas y aportaciones fueron escuchadas y valoradas por el equipo?



Ilustración 136 - Pregunta 3 formulario de equipo

¿Qué tanto te sentiste motivado/a y comprometido/a a lo largo del proyecto?



Ilustración 137 - Pregunta 4 formulario de equipo

¿Cómo calificarías la capacidad del equipo para resolver problemas o superar desafíos durante el proyecto?

1 2 3 4 5

Muy mal

Excelente

Ilustración 138 - Pregunta 5 formulario de equipo

Los resultados obtenidos fueron los siguientes:

¿Cómo calificarías el nivel de apoyo y ayuda que recibiste de los otros miembros del equipo?

Copiar

3 respuestas

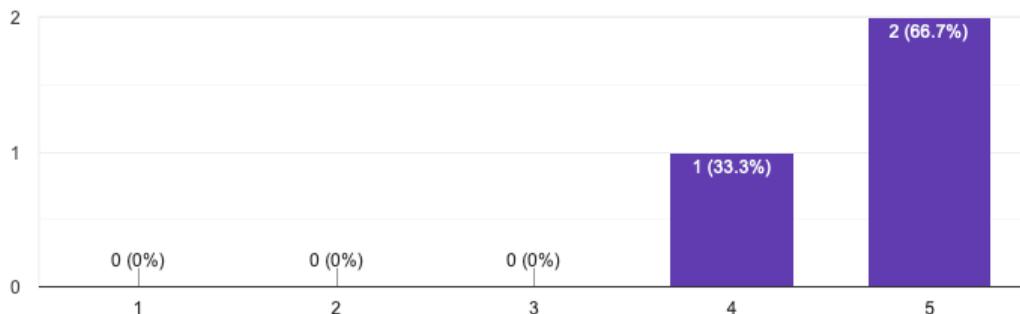


Ilustración 139 - Resultado pregunta 1

¿Qué tan clara fue la asignación de roles y responsabilidades dentro del equipo?

 Copiar

3 respuestas

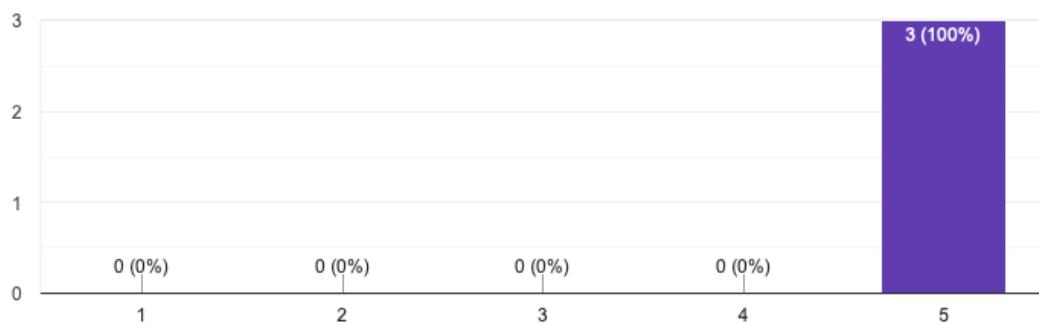


Ilustración 140 - Resultado pregunta 2

¿En qué medida sentiste que tus ideas y aportaciones fueron escuchadas y valoradas por el equipo?

 Copiar

3 respuestas

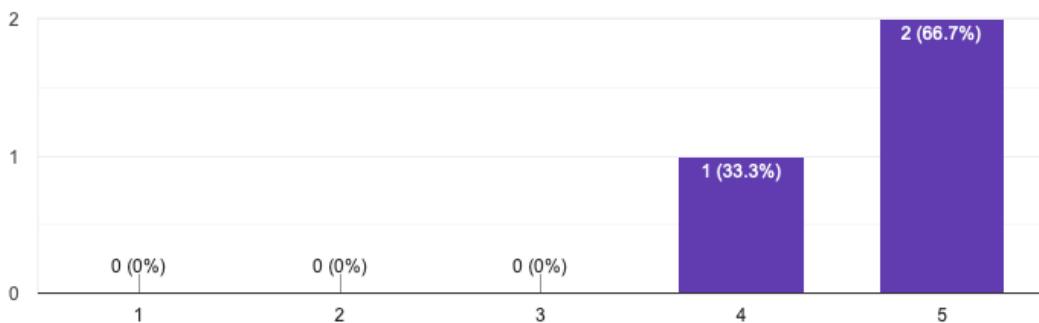


Ilustración 141 - Resultado pregunta 3

¿Qué tanto te sentiste motivado/a y comprometido/a a lo largo del proyecto?

 Copiar

3 respuestas

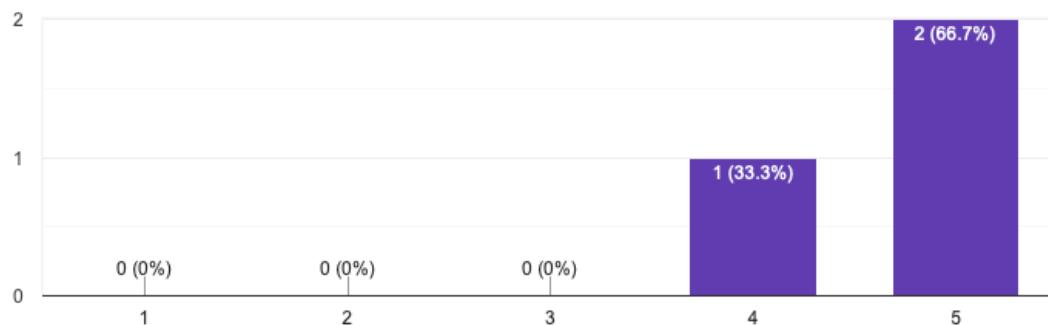


Ilustración 142 - Resultado pregunta 4

¿Cómo calificarías la capacidad del equipo para resolver problemas o superar desafíos durante el proyecto?

 Copiar

3 respuestas

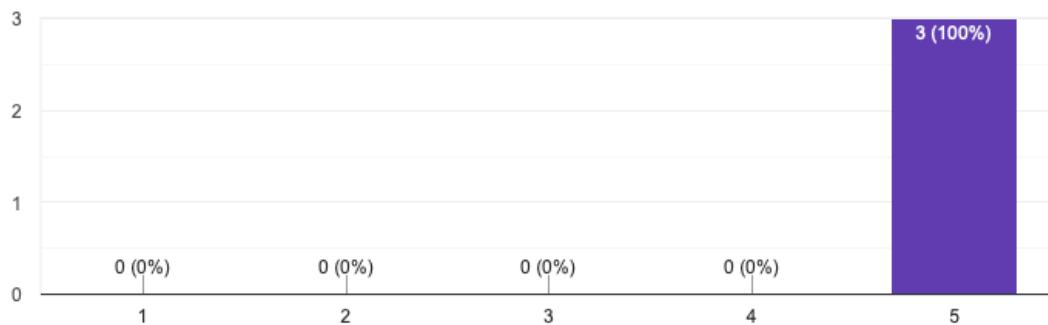


Ilustración 143 - Resultado pregunta 5