

UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA  
DEPARTAMENTO DE INFORMÁTICA  
VALPARAÍSO - CHILE



“ADOPCIÓN DE PRÁCTICAS ÁGILES PARA LA  
IMPLEMENTACIÓN DE UN PUNTO VENTA ENFOCADO  
EN UNA EMPRESA PEQUEÑA DEDICADA A LA  
GASTRONOMÍA”

YI CONG ZHOU ZHOU

MEMORIA PARA OPTAR AL TÍTULO DE  
INGENIERO CIVIL EN INFORMÁTICA

Profesor Guía: Wladimir Elías Ormazábal Orellana  
Profesor Correferente: Pedro Francisco Godoy Barrera

Agosto - 2025



## CONSTANCIA DE VALIDACIÓN Y CONFIDENCIALIDAD DE MONOGRAFÍA A REPOSITORIO ACADÉMICO

### 1.- IDENTIFICACIÓN DEL TRABAJO ACADÉMICO

Tipo de monografía (marcar una opción): ☒ Memoria o trabajo de título; ☐ Tesis de Postgrado;

Título del trabajo: "ADOPCIÓN DE PRÁCTICAS ÁGILES PARA LA IMPLEMENTACIÓN DE UN PUNTO DE VENTA ENFOCADO EN UNA EMPRESA PEQUEÑA DEDICADA A LA GASTRONOMÍA"

Nombre del candidato(a): Yi Cong Zhou Zhou

Carrera / Grado: Ingeniería Civil Informática

Campus: Casa Central Valparaíso ; Departamento: Informática

### 2.- VALIDACIÓN DEL PROFESOR GUÍA/DIRECTOR DE TESIS

Yo, Wladimir Elías Ormazábal Orellana, en mi calidad de profesor(a) guía/director(a) del trabajo académico mencionado anteriormente **DEJO CONSTANCIA** que:

- He revisado esta versión del documento y corresponde a la versión final aprobada del trabajo.
- El trabajo cumple con los requisitos académicos y de formato establecidos por la institución

### 3.- EVALUACIÓN DE CONFIDENCIALIDAD POR PROPIEDAD INDUSTRIAL

☒ El trabajo **NO** contiene información que amerite confidencialidad y puede ser publicado de inmediato en repositorio con acceso abierto.

☐ El trabajo **CONTIENE** información con potenciales implicancias de propiedad industrial o intelectual y requiere un periodo de confidencialidad (embargo) por:

☐ 6 meses; ☐ 12 meses; ☐ 2 años; ☐ 3 años; ☐ 5 años; ☐ 10

años Fundamentación de la necesidad de confidencialidad (obligatorio si se solicita embargo):

### 4.- FIRMAS

Profesor(a) guía o director(a) de memoria o tesis:

Fecha: 30-07-2025 ; Firma:

Estudiante o Candidato(a):

Fecha: 30-07-2025 ; Firma:

*Este formulario debe ser insertado como página 2 de la memoria o tesis, completado y firmado por estudiante y profesor(a) antes de la entrega en portal PRISMA de Biblioteca USM.*

## AGRADECIMIENTOS

A mis padres, que tuvieron que hacer el sacrificio 35 años atrás de migrar a Chile, sin saber nada del idioma ni las costumbres de este país. Les agradezco todos los días que tuvieron que esforzarse para que mi hermano y yo pudiéramos tener todas las comodidades y necesidades que se nos presentaban.

A mi hermano, quien era prácticamente la única persona con quien jugaba hasta que entré al colegio, con el vi un montón de series, jugué clásicos y copié gran parte de las cosas que hacía.

A mis amigos de la universidad, Axel, Benjamín, Braulio, Claudio, Javier, Jesús, Joaquín, Pedro y Vicente, por formar parte de este proceso. Con ellos viví un montón de experiencias dentro de la universidad; las sesiones en discord viendo las votaciones de los paros estudiantiles, las sesiones en discord en pandemia, las idas a la biblioteca y al DI cuando se acercaban las fechas de los certámenes, los carretes post-certámenes y los partidos de volleyball. Con ellos me formé como profesional y les agradezco a cada uno de ellos por formar parte de mi vida estudiantil.

A mis amigos de la infancia, en especial Iván y Tomás, quienes me acompañaron desde Kinder hasta cuarto medio. Les agradezco por ser aquellas personas con quienes puedo volver a ser un niño y despejarme en ocasiones en donde me siento decaído.

Por último, al profesor Wladimir por aceptar mi propuesta de tema y por acompañarse durante este proceso, le agradezco su paciencia y su buena voluntad de guiarme y acompañarme durante el desarrollo de mi memoria.

## RESUMEN

**Resumen**— Un Punto de Venta (POS) resulta fundamental para un negocio de servicios alimentarios, ya que aporta eficiencia y competitividad. Aunque el enfoque ágil parece ser la opción idónea para su desarrollo, su implementación puede ser costosa para pequeñas empresas con presupuestos ajustados o que no planean invertir una gran cantidad de recursos en ello. Por consiguiente, en el presente documento se procederá a analizar, seleccionar y utilizar cuidadosamente prácticas de desarrollo ágil que se adapten a las limitaciones financieras y temporales de un negocio enfocado en servicios alimentarios, sin comprometer la calidad del producto final.

**Palabras Clave**— Punto de Venta (POS); enfoque ágil; pequeñas empresas.

## ABSTRACT

**Abstract**— A Point of Sale (POS) system is essential for a food service business as it enhances efficiency and competitiveness. Although the agile approach appears to be the ideal option for its development, its implementation can be costly for small businesses with tight budgets or those that do not plan to invest a large amount of resources in it. Therefore, this document will proceed to analyze, select, and carefully use agile development practices that fit the financial and time constraints of a business focused on food services, without compromising the quality of the final product.

**Keywords**— Point of Sale (POS); agile approach; small businesses.

## GLOSARIO

API: Interfaz de programación de aplicaciones.

CA: Criterio de aceptación.

Comanda: Documento que se utilizan los garzones para registrar los productos de los clientes.

Diseño responsivo: Técnica de diseño que permite redimensionar los elementos de una página web o aplicación conforme se vaya aumentando o disminuyendo el tamaño de la ventana.

Framework: Entorno de trabajo o marco de trabajo. Dentro de este contexto, el framework entregará módulos, bibliotecas y/o programas para facilitar el desarrollo del producto.

Frontend: Parte de una aplicación o sistema web en la cual el usuario puede ver contenido y realizar acciones.

HU: Historia de usuario.

IDE: Entorno de desarrollo integrado.

Layout: Disposición o estructura visual en donde se organizan y distribuyen componentes físicos o digitales.

OE: Objetivo específico.

PA: Práctica ágil.

PK: Llave primaria.

POS: Punto de venta.

Voucher: Comprobante de venta de algún producto.

XP: Extreme Programming.

# ÍNDICE DE CONTENIDOS

|                                      |     |
|--------------------------------------|-----|
| RESUMEN                              | II  |
| ABSTRACT                             | II  |
| GLOSARIO                             | III |
| ÍNDICE DE FIGURAS                    | VI  |
| ÍNDICE DE TABLAS                     | VII |
| INTRODUCCIÓN                         | 1   |
| CAPÍTULO 1: DEFINICIÓN DEL PROBLEMA  | 3   |
| 1.1 Contexto Preliminar              | 3   |
| 1.2 Identificación del problema      | 4   |
| 1.3 Objetivos                        | 6   |
| 1.3.1 Objetivo General               | 6   |
| 1.3.2 Objetivos Específicos          | 7   |
| CAPÍTULO 2: MARCO CONCEPTUAL         | 8   |
| 2.1 Punto de venta                   | 8   |
| 2.2 Ingeniería de Software           | 9   |
| 2.3 Desarrollo de software           | 10  |
| 2.4 Metodología ágil                 | 10  |
| 2.4.1 SCRUM                          | 11  |
| 2.4.2 Kanban                         | 13  |
| 2.4.3 Scrumban                       | 14  |
| 2.4.4 Extreme Programming            | 14  |
| CAPÍTULO 3: PROPUESTA DE SOLUCIÓN    | 16  |
| 3.1 Prácticas a utilizar             | 16  |
| 3.1.1 Kanban Board y Product Backlog | 16  |
| 3.1.2 Limitar el trabajo             | 17  |
| 3.1.3 Ritmo sostenible               | 17  |
| 3.1.4 Daily standup                  | 17  |
| 3.1.5 Iteraciones                    | 18  |
| 3.1.6 Estandarización de código      | 18  |
| 3.2 Introducción del negocio         | 25  |
| 3.2.1 Actores                        | 29  |
| 3.3 Elección de tecnologías          | 31  |
| 3.3.1 Frontend                       | 31  |
| 3.3.2 Base de datos                  | 33  |

|                                                |    |
|------------------------------------------------|----|
| 3.3.3 Entorno de desarrollo integrado (IDE)    | 33 |
| 3.4 Especificación del software                | 34 |
| 3.5 Estimación de esfuerzo                     | 34 |
| 3.6 Estimación de prioridad                    | 36 |
| 3.7 Historias de usuario                       | 37 |
| 3.7.1 HU1                                      | 37 |
| 3.7.2 HU2                                      | 37 |
| 3.7.3 HU3                                      | 38 |
| 3.7.4 HU4                                      | 39 |
| 3.7.5 HU5                                      | 40 |
| 3.7.6 HU6                                      | 40 |
| 3.7.7 HU7                                      | 41 |
| 3.7.8 HU8                                      | 42 |
| 3.7.9 HU9                                      | 42 |
| 3.7.10 HU10                                    | 43 |
| 3.8 Organización de iteraciones                | 44 |
| 3.8.1 Primera iteración                        | 45 |
| 3.8.2 Segunda iteración                        | 47 |
| 3.8.3 Tercera iteración                        | 49 |
| 3.8.4 Cuarta iteración                         | 51 |
| CAPÍTULO 4: VALIDACIÓN DE LA SOLUCIÓN          | 55 |
| 4.1 Vistas del software                        | 55 |
| 4.1.1 Inicio de sesión                         | 55 |
| 4.1.2 Dashboard                                | 56 |
| 4.1.3 Ventas                                   | 57 |
| 4.1.4 Inventario                               | 59 |
| 4.1.5 Historial                                | 60 |
| 4.2 Periodo de prueba                          | 61 |
| 4.3 Prueba de estrés                           | 63 |
| 4.4 Impacto del sistema generado en el negocio | 65 |
| 4.5 Impacto de las prácticas ágiles            | 72 |
| CAPÍTULO 5: CONCLUSIONES                       | 75 |
| REFERENCIAS BIBLIOGRÁFICAS                     | 78 |

## ÍNDICE DE FIGURAS

|    |                                                                           |    |
|----|---------------------------------------------------------------------------|----|
| 1  | Distribución de metodologías ágiles en empresas encuestadas.              | 11 |
| 2  | Ejemplo tablero Kanban.                                                   | 17 |
| 3  | Vista actual de la sección de Cronología dentro de Journey.               | 18 |
| 4  | Configuración en la sección "Code Style/Wrapping and Braces" del IDE.     | 25 |
| 5  | Visualización toma de pedidos con servicio dentro del restaurante.        | 26 |
| 6  | Diagrama de carril. Visualización toma de pedidos para llevar.            | 27 |
| 7  | Diagrama de carril. Visualización toma de pedidos para envío a domicilio. | 28 |
| 8  | Diagrama de contexto del sistema POS.                                     | 30 |
| 9  | Variación de la matriz de Eisenhower.                                     | 36 |
| 10 | Vista del Product backlog.                                                | 45 |
| 11 | Vista del tablero durante la primera iteración. 05 de Junio.              | 46 |
| 12 | Vista del tablero durante la segunda iteración. 18 de Junio.              | 48 |
| 13 | Vista del tablero durante la tercera iteración. 19 de Julio.              | 50 |
| 14 | Vista del tablero durante la cuarta iteración. 02 de Agosto.              | 53 |
| 15 | Vista del software, sección de login.                                     | 55 |
| 16 | Vista del software, sección de dashboard.                                 | 57 |
| 17 | Vista del software, sección de ventas para llevar y mesas.                | 58 |
| 18 | Vista del software, sección de una venta para mesa en específico.         | 58 |
| 19 | Vista del software, sección de una venta para llevar en específico.       | 59 |
| 20 | Vista del software, sección de inventario.                                | 60 |
| 21 | Vista del software, sección de historial.                                 | 61 |
| 22 | Gráfico comparativo rendimiento vs tiempo.                                | 64 |
| 23 | Tiempo de registro por pedido utilizando el sistema POS.                  | 68 |

|                                                                             |    |
|-----------------------------------------------------------------------------|----|
| 24 Diagramas de caja tiempo de registro por cantidad de productos. . . . .  | 69 |
| 25 Mejora en tiempo de registro según cantidad de productos únicos. . . . . | 71 |

## ÍNDICE DE TABLAS

|                                                                                                                            |    |
|----------------------------------------------------------------------------------------------------------------------------|----|
| 1 Prácticas comunes utilizadas en metodologías ágiles. . . . .                                                             | 5  |
| 2 Contenidos de la ingeniería de software. . . . .                                                                         | 9  |
| 3 Ventajas y desventajas de cada lenguaje de programación. . . . .                                                         | 31 |
| 4 Precios de hosting. . . . .                                                                                              | 32 |
| 5 Precios de dominios .cl. . . . .                                                                                         | 32 |
| 6 Ejemplo de valores de historias de usuario. . . . .                                                                      | 35 |
| 7 Especificaciones técnicas del entorno tecnológico utilizado. . . . .                                                     | 61 |
| 8 Monitoreo del funcionamiento del software en una etapa temprana, periodo<br>02 de septiembre - 08 de septiembre. . . . . | 62 |
| 9 Horarios de funcionamiento del local estudiado. . . . .                                                                  | 63 |
| 10 Iteraciones realizadas durante el experimento. . . . .                                                                  | 63 |
| 11 Simulación hoja de detalle utilizada anteriormente dentro del negocio. . . . .                                          | 65 |
| 12 Tiempos de registro manual de ventas según la cantidad de productos. . . . .                                            | 67 |
| 13 Tiempos de registro mediante el uso del sistema POS según cantidad de<br>productos. . . . .                             | 69 |

## INTRODUCCIÓN

La industria gastronómica enfrenta diversos desafíos en su operación diaria, especialmente en lo que respecta a la eficiencia de la atención al cliente y la administración interna. En este contexto, los sistemas de Punto de Venta (POS) se han consolidado como herramientas clave para agilizar procesos como la toma de pedidos, el cobro, la gestión de inventario, seguimiento de pedidos y la obtención de métricas de rendimiento. Estos sistemas permiten una mejora considerable en la organización y trazabilidad de las operaciones, disminuyendo los errores humanos y optimizando los tiempos de atención.

Sin embargo, la implementación de un sistema POS no siempre es una tarea sencilla para pequeñas empresas del rubro gastronómico. Factores como presupuestos acotados, la poca o nula experiencia en procesos formales de levantamiento de requisitos y la falta de personal técnico dificultan la adopción de soluciones tecnológicas sofisticadas. En este escenario, el uso de prácticas ágiles se presenta como una alternativa prometedora, ya que permite una adaptación continua a los requerimientos del negocio, siempre que estas se ajusten de forma realista a las restricciones del proyecto.

El presente proyecto tiene como propósito estudiar y aplicar un conjunto acotado de prácticas asociadas a metodologías ágiles, con el objetivo de flexibilizar el desarrollo de un sistema POS en un escenario con restricciones temporales, económicas y con un grado de incertidumbre respecto a la evolución de sus requerimientos, sin comprometer la calidad del software.

Para lograr este objetivo, se analizará el contexto del negocio y sus limitaciones administrativas y estructurales, se seleccionarán prácticas ágiles relevantes para el contexto de desarrollo, se definirá un conjunto de funcionalidades por realizar y se realizará un seguimiento del impacto de estas prácticas durante el proceso de desarrollo. Además, se documentará el proceso de implementación, incluyendo decisiones tecnológicas, organización del trabajo y se validarán las funcionalidades desarrolladas.

Este documento se estructura de la siguiente manera: en el capítulo 1, se presenta la definición del problema, en donde se introduce el contexto preliminar e identifican los problemas asociados al desarrollo del sistema POS en un contexto microempresarial, además de establecer los objetivos del presente proyecto. En el capítulo 2, se define el marco conceptual, en donde se introduce la ingeniería y desarrollo de software, además de las metodologías más utilizadas de esta última y las prácticas que cada una de estas emplea. En el capítulo 3, se presenta la propuesta de solución, donde se definen las prácticas ágiles a aplicar e introducir brevemente el negocio, para luego elegir un conjunto de tecnologías a utilizar en base a las ventajas comparativas que tienen y los enfoques que ambos presentan: aplicación de escritorio o web. Por otra parte, también se detallan las funcionalidades consideradas, finalizando con la organización del proceso de desarrollo. En el capítulo 4, se muestran los resultados obtenidos a través de un periodo de prueba; las vistas desarrolladas, métricas relacionadas al uso del software, el impacto que tuvieron las prácticas ágiles utilizadas, el

comportamiento del sistema desarrollado en condiciones de alta demanda y el impacto que este genera dentro del negocio. Finalmente, en el capítulo 5 se presentan las conclusiones de este documento, detallando el grado de cumplimiento de los objetivos propuestos, las lecciones aprendidas durante el proceso de desarrollo y posibles mejoras o ramificaciones que podría tener el software implementado.

## CAPÍTULO 1

### DEFINICIÓN DEL PROBLEMA

#### 1.1. Contexto Preliminar

Dentro del ámbito de los restaurantes, la adopción de un sistema de punto de venta (POS) se ha convertido en una operación fundamental para una gestión eficiente. Estos sistemas no solo reemplazan el uso del lápiz y papel a la hora de realizar las comandas, sino que también facilitan el proceso de toma de pedidos y pagos e integran una gran variedad de herramientas y funciones diseñadas para optimizar todas las operaciones, desde la gestión de inventario hasta el análisis de datos de ventas. El uso de dichos sistemas permite una mayor agilidad en el servicio y reducir errores que se podrían generar en las operaciones manuales, como las que se listan a continuación:

- Ilegibilidad de la comanda.
- Pérdida de comandas.
- Pérdida de tiempo cuando algún cliente pide dividir la cuenta o cuando se requiere hacer algún tipo de descuento.
- Mayor riesgo de cometer algún error al realizar la cuenta (esta probabilidad de error se incrementa mientras mayor sea la cantidad de productos pedidos).
- Aumento del tiempo necesario para realizar la cuenta.
- Dificultad de obtener indicadores de ganancia de manera sencilla.

Es por esto que, al automatizar dichos procesos a través de estos tipos de sistemas, se agiliza aún más el flujo de trabajo, lo que se traduce en tiempos de espera reducidos y una mayor satisfacción del cliente.

Otro beneficio significativo es su capacidad para recopilar y analizar datos en tiempo real. Al rastrear cada transacción y recopilar información sobre todos los pedidos realizados, los restaurantes pueden obtener una visión profunda de su rendimiento y ajustar su estrategia en consecuencia. A través de esto, se pueden obtener diversas métricas que pueden ser útiles para el desarrollo y crecimiento de un negocio. A continuación, se listan algunas métricas que se pueden obtener:

- Cantidad de pedidos realizados en un día y ver en qué hora hay mayor concentración de pedidos u hora punta, con lo cuál el negocio sería capaz de optimizar sus productos y su mano de obra en base a alguna hora y día específico.

- Cantidad de veces que se vende algún producto, pudiendo identificar algún alimento que no esté siendo rentable y realizar modificaciones pertinentes.
- Monto promedio de venta, es decir, el valor promedio que nos otorgaría cada cliente atendido.
- Tiempo promedio de servicio de cada mesa, el cual nos ayudaría a entender cuánto tiempo gasta en promedio cada cliente dentro del negocio.

Desarrollar este tipo de sistema de manera personalizada utilizando una metodología ágil de desarrollo de software puede ser una opción viable en una primera instancia, especialmente para aquellos que desean funcionalidades específicas adaptadas a sus necesidades únicas. Estas prácticas ágiles permiten una flexibilidad en el proceso de desarrollo, lo que significa que se pueden realizar ajustes sobre la marcha según los requisitos cambiantes del negocio o la retroalimentación brindada por el cliente.

## 1.2. Identificación del problema

Si bien se mencionó anteriormente que sería tentativo utilizar metodologías ágiles para la realización de un sistema POS, emplear completamente un enfoque ágil podría no ser la mejor opción para pequeñas empresas que no pueden adecuarse a todas las prácticas propias y desafíos inherentes de los enfoques ágiles. Por ejemplo, en [1] se entrevistaron a 16 organizaciones enfocadas al desarrollo de software, en donde uno de los principales desafíos era la estimación tanto de costos como la duración del proyecto, teniendo que, en algunos casos, modificar de manera muy dramática la estimación durante el desarrollo. Estos desafíos pueden no ser un impedimento para empresas establecidas que cuentan con flexibilidad tanto monetaria como de tiempo, pero puede no ser el caso para microempresas que cuentan con una cantidad fija de recursos para invertir y una urgente necesidad de adquirir un software que se adecúe a sus requerimientos.

Otra problemática a la hora de adoptar un método ágil es tener en consideración el tamaño del equipo, esto debido a que ciertas prácticas de algún método podrían no tener sentido dependiendo de cuántas personas estén desarrollando el proyecto. A continuación, se presentará una tabla en donde se detallan algunas prácticas comunes de métodos ágiles.

| Práctica ágil                             | Nombres adicionales y prácticas ágiles similares                                                                                                                                                                                                                                                                                  |
|-------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| PA1: Equipo co-localizado                 | Espacio/área de oficina abierta, espacio de oficina co-localizado, entorno de trabajo abierto, conversaciones de tipo cara a cara dentro del entorno de trabajo, espacios de trabajo abiertos.                                                                                                                                    |
| PA2: Involucramiento del cliente          | Miembro del equipo del cliente, iteraciones dirigidas por el cliente, participación directa del usuario final, cliente in-situ, colaboración con el cliente.                                                                                                                                                                      |
| PA3: Equipo auto-organizado               | Autonomía del equipo.                                                                                                                                                                                                                                                                                                             |
| PA4: Visualización continua del progreso  | Seguimiento del progreso, visualización continua, tablero de tareas, tablero de estado, tablero de información, espacio de trabajo informativo.                                                                                                                                                                                   |
| PA5: Entregas incrementales               | Entregas frecuentes, revisión de entregas, lanzamientos tempranos, retroalimentación ágil, revisión del sprint, demostración, retroalimentación del cliente, revisión de iteraciones, reuniones colectivas / de revisión                                                                                                          |
| PA6: Documentación mínima                 | Documentación ágil                                                                                                                                                                                                                                                                                                                |
| PA7: Retrospectiva                        | Reuniones retrospectivas, retrospección de la iteración, taller de reflexión, talleres post-iteración, retrospectiva del sprint, reuniones retrospectivas colectivas.                                                                                                                                                             |
| PA8: Requerimientos priorizados por valor | Juegos de planificación, simplicidad en el diseño, especificación de requerimientos, priorización de requerimientos, historias de usuario, iteraciones dirigidas por el cliente, backlog de iteraciones, backlog del producto, backlog del sprint, metáfora, especificación ágil, recopilación y clarificación de requerimientos. |
| PA9: Integración continua del código      | Integración frecuente, compilaciones continuas, compilaciones automatizadas, integración continua.                                                                                                                                                                                                                                |
| PA10: Programación en pares               | No se identifican nombres adicionales ni prácticas ágiles similares.                                                                                                                                                                                                                                                              |
| PA11: Iteraciones cortas (Sprints)        | Ciclos iterativos cortos, planificación de iteraciones (reuniones), planificación constante, iteraciones ágiles.                                                                                                                                                                                                                  |
| PA12: Reuniones en pie                    | Seguimiento diario del progreso, reuniones diarias (del equipo), reuniones diarias en pie, reunión scrum.                                                                                                                                                                                                                         |

A modo de ejemplo, si el proyecto es gestionado por un solo desarrollador o un equipo pequeño, la implementación de ciertas prácticas de metodologías ágiles puede no ser muy eficaz. Por ejemplo, el *Pair Programming*, propio de la metodología *Extreme Programming*, no tendría sentido cuando una sola persona está a cargo del desarrollo, ya que esta técnica se basa en la colaboración continua entre dos programadores. De igual manera, realizar *Stand-up meetings*, una práctica común en *SCRUM*, podría ser innecesario en un equipo muy pequeño o en un proyecto con un único encargado, ya que estas reuniones buscan la comunicación eficiente entre varios integrantes, dando a conocer a los demás integrantes en qué se ha trabajado, qué se realizará y qué obstáculos ha enfrentado cada uno. Por otro lado, si el equipo fuera extremadamente grande, la duración y efectividad de las *Stand-up meetings* se verían comprometidas, diluyendo su propósito de sintetizar las labores realizadas y por realizar en un corto tiempo.

Adicionalmente, otro desafío a tener en consideración son las barreras [3] a la hora de involucrar al cliente en la toma de requisitos y en la retroalimentación. Es muy probable que muchos de estos micronegocios no tengan experiencia o tengan poca experiencia en todo lo que conlleva un proceso de levantamiento de requerimientos, como la falta de conocimiento técnico y de terminología afín a estos procesos (por ejemplo, historias de usuario, casos de uso, mockups, etc.), la dificultad para traducir sus ideas a un entorno informático y las expectativas inciertas. Esta falta de experiencia puede llegar a entorpecer la toma de requerimientos y la retroalimentación dada por el cliente, lo que podría concluir en una demora en el desarrollo del sistema. Debido a esto, es fundamental definir estrategias que permitan una comunicación periódica y la capacidad de entender y modelar dichas necesidades en requisitos de manera correcta, con el fin de cimentar bases sólidas sobre las cuales se trabajará.

### 1.3. Objetivos

Tomando en consideración el contexto y los problemas mencionados anteriormente, dentro de este escrito se busca desarrollar un software que se acomode a las necesidades presentadas por la empresa, con el fin de agilizar y automatizar procesos administrativos.

#### 1.3.1. Objetivo General

**“Adopción de prácticas ágiles en el contexto de desarrollo de un sistema POS tomando como base presiones monetarias y de tiempo.”**

### **1.3.2. Objetivos Específicos**

- **OE1: Buscar, estudiar y determinar el impacto de aplicar prácticas comúnmente asociadas a metodologías ágiles.**
- **OE2: Evaluar ventajas y desventajas de dos enfoques definidos a priori: Aplicación de escritorio y sistema web, para tomar decisiones costo-efectivas según el contexto del proyecto.**
- **OE3: Monitoreo del software en etapas tempranas de su implementación, para conocer su impacto y adopción.**

## CAPÍTULO 2

### MARCO CONCEPTUAL

#### 2.1. Punto de venta

Un punto de venta es un sistema informático que se utiliza en los negocios minoristas y de servicio para realizar transacciones de venta. Su función principal es procesar pagos de clientes por bienes o servicios y concentrar en un único sistema todas las funcionalidades administrativas de un negocio, a modo de agilizar sus procesos y toma de decisiones.

A continuación, se detallan algunas características que puede llegar a tener en base al tipo de negocio y sus necesidades.

- Registro de transacciones: Permite registrar y procesar transacciones de venta, incluyendo la selección de productos o servicios, el cálculo del total de la compra, la aplicación de descuentos, impuestos y otros cargos.
- Gestión de inventario: Ayuda a gestionar el inventario mediante el seguimiento de las existencias de productos. Puede proporcionar alertas de inventario bajo, realizar pedidos automáticos y mantener actualizada la información sobre la disponibilidad de productos.
- Procesamiento de pagos: Facilita el procesamiento de pagos a través de diversos métodos, como efectivo, tarjetas de crédito o débito y pagos móviles. También, es posible utilizar un terminal de pago externo para la transacción, como por ejemplo Transbank o Redelcom.
- Gestión de clientes: Almacena información sobre los clientes, como datos de contacto, historial de compras y preferencias. Esto permite ofrecer programas de fidelización, promociones personalizadas y mejorar la atención al cliente.
- Generación de informes: Ofrece la capacidad de entregar información relevante para el dueño del negocio. Usualmente, esa información es ilustrada en un dashboard en donde se muestran diversos indicadores de ganancia.
- Personalización y configuración: Permite personalizar el sistema según las necesidades específicas del negocio, como la configuración de precios, la creación de promociones y la adaptación de la interfaz de usuario.

## 2.2. Ingeniería de Software

El término *ingeniería de software* surgió a inicios de los años 60 durante la denominada *Crisis del software* [4]. Esta crisis, caracterizada por proyectos que excedían presupuestos y plazos, así como por sistemas que no cumplían con los requisitos mínimos de calidad y funcionalidad, evidenció la necesidad de establecer reglas y metodologías que permitieran abordar los crecientes desafíos en el diseño, desarrollo y mantenimiento del software.

Como resultado, en la actualidad la ingeniería de software se reconoce como una disciplina de la informática enfocada en la aplicación de principios de la ingeniería a todos los procesos relacionados con la creación, gestión y mantenimiento de software. En la [Tabla 2](#) se presenta un diagrama con todas las aristas de esta disciplina, las cuales son presentadas y actualizadas por la *IEEE Computer Society* dentro del documento *Software Engineering Body of Knowledge (SWEBOK)* [5] y que tienen como función asegurar que los productos finales sean confiables, eficientes y cumplan con las necesidades del cliente.

| Nº | Categoría                |
|----|--------------------------|
| 1  | Requerimientos           |
| 2  | Arquitectura             |
| 3  | Diseño                   |
| 4  | Construcción             |
| 5  | Pruebas                  |
| 6  | Operaciones              |
| 7  | Mantenición              |
| 8  | Gestión de configuración |
| 9  | Gestión                  |
| 10 | Procesos                 |
| 11 | Modelos y métodos        |
| 12 | Calidad                  |
| 13 | Seguridad                |
| 14 | Prácticas profesionales  |
| 15 | Economía                 |
| 16 | Bases de la computación  |
| 17 | Bases matemáticas        |
| 18 | Bases de la ingeniería   |

Tabla 2: Contenidos de la ingeniería de software.  
Fuente: Elaboración propia.

## 2.3. Desarrollo de software

Por su parte, el desarrollo de software es un subconjunto más específico que es fundamental en el área de la informática y la ingeniería de software. Este proceso, como indica su nombre, se centra en la escritura, prueba y mantenimiento del código a través de algún lenguaje de programación. Este proceso se puede realizar a través de diversas metodologías, dentro de las cuales se seleccionó la metodología ágil como método de estudio.

## 2.4. Metodología ágil

En los inicios del desarrollo de software, la programación se realizaba sin procesos estructurados, lo que dificultaba la planificación y mantenimiento de los sistemas. Para abordar estos problemas, durante los años 70 surgió el modelo en cascada, un enfoque lineal en el que el desarrollo se dividía en fases secuenciales: requisitos, diseño, implementación, pruebas y mantenimiento. Si bien este modelo permitía una planificación detallada, este enfoque presenta varias desventajas significativas. Por ejemplo, al requerir una definición exhaustiva de los requisitos al inicio del proyecto, se vuelve difícil anticipar todos los aspectos del producto final, lo que puede resultar en costosos cambios y retrasos si surgen nuevas necesidades o problemas durante el desarrollo. Para emplear esta metodología de manera efectiva, se requiere un conocimiento exhaustivo del dominio y un entorno estable, lo que permite levantar requerimientos que no variarán con el tiempo. En contextos donde los requisitos puedan cambiar o evolucionar, como el del presente documento, el enfoque en cascada podría conducir al desarrollo de un software que no cumpla con las expectativas o necesidades del cliente, debido a la limitada flexibilidad y comunicación durante el proceso.

Debido a las desventajas mencionadas anteriormente, un conjunto de desarrolladores propuso 12 principios en el denominado *Manifiesto ágil* [6]. Estos principios se enfocaban principalmente en satisfacer al cliente mediante entregas tempranas y continuas, fomentar la comunicación entre los equipos de desarrollo y conformar equipos auto-organizados, con lo que se formaría la denominada metodología ágil.

Esta metodología es un enfoque flexible y colaborativo para el desarrollo de software, el cual se centra en la adaptabilidad, la colaboración continua y la entrega iterativa de software funcional. Sus principales características incluyen la priorización de la satisfacción del cliente, la entrega frecuente de productos útiles, la colaboración cercana entre los equipos multidisciplinarios y la capacidad de respuesta al cambio, incluso en etapas avanzadas del desarrollo. En lugar de seguir un plan detallado desde el principio como el método cascada, los equipos ágiles responden a los cambios y retroalimentaciones a medida que avanzan, lo que les permite adaptarse rápidamente a las necesidades del cliente y a los desafíos del proyecto. Cabe destacar que estos cambios deben realizarse de manera clara y controlada, llegando a un acuerdo entre el equipo de desarrollo y la parte interesada.

Durante la última década, ha habido un marcado aumento en la adopción de metodologías ágiles por parte de los equipos de desarrollo. Este cambio se refleja claramente en un estudio realizado por *Digital.ai* durante el año 2021 [7], en donde se menciona un incremento significativo en la implementación de estas prácticas. Según la encuesta realizada a empresas dedicadas a los diversos rubros de la informática, la adopción de metodologías ágiles pasó de un 37 % en el año 2020 a un 84 % en el año 2021. Este aumento refleja la importancia de este tipo de metodologías y es una de las razones del porqué se empleará durante el transcurso de este documento.

A continuación, y haciendo uso del mismo estudio, se muestra en la **Figura 1** la distribución de las metodologías ágiles más utilizadas, las cuales se definirán y se darán a conocer las prácticas más comunes de cada una a lo largo de este capítulo.

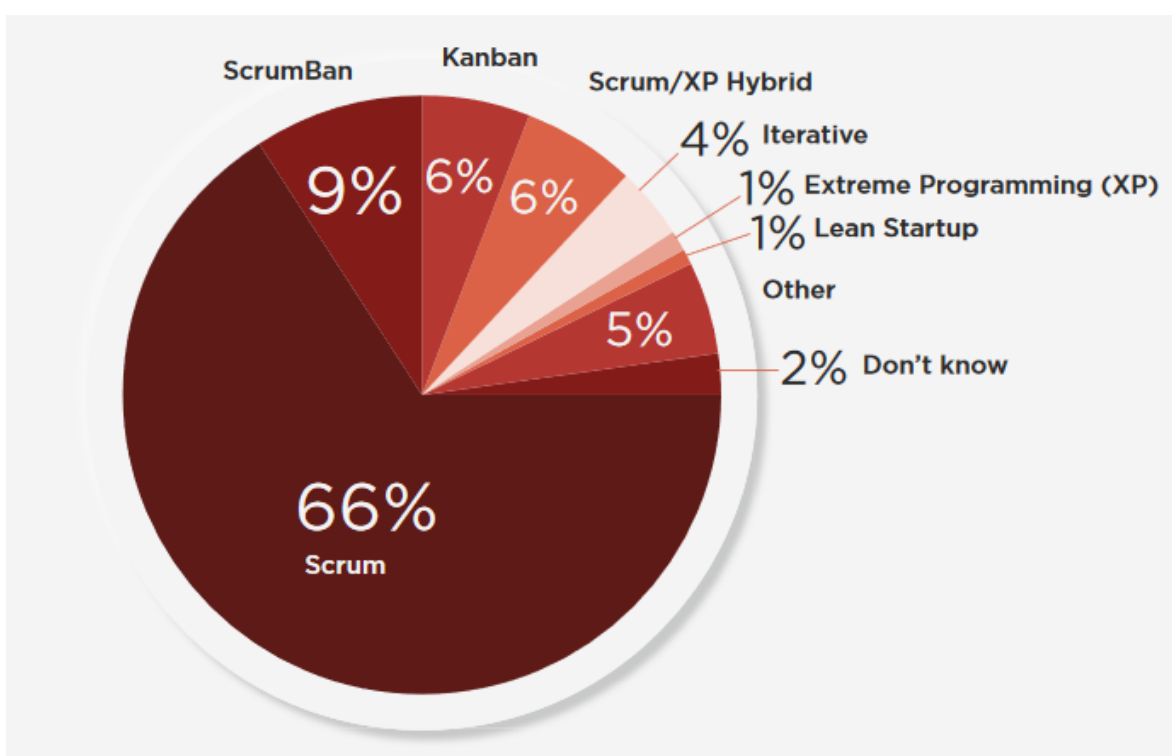


Figura 1: Distribución de metodologías ágiles en empresas encuestadas.

Fuente: [7].

#### 2.4.1. SCRUM

Scrum es un marco de trabajo ágil ampliamente utilizado en el desarrollo de productos complejos. Se caracteriza por su enfoque en la transparencia, la inspección y la adaptación. En Scrum, el trabajo se organiza en intervalos de tiempo llamados sprints, que suelen tener una duración de una a cuatro semanas. Durante cada sprint, el equipo se enfoca en completar un

conjunto de funcionalidades prioritarias, conocido como sprint backlog. Los equipos de desarrollo se autoorganizan y colaboran estrechamente para cumplir con los objetivos establecidos. Al final de cada sprint, se lleva a cabo una revisión para demostrar el trabajo realizado y recibir comentarios del cliente o usuarios finales. También se realiza una retrospectiva para identificar áreas de mejora y ajustar el proceso para el próximo sprint.

Scrum define roles específicos, como el Scrum Master, responsable de facilitar el proceso, y el Product Owner, encargado de definir y priorizar el backlog del producto.

### Prácticas comunes de SCRUM

- **Roles:** La división de roles del equipo de desarrollo está dividido principalmente en tres roles; Product Owner, Scrum Master y Desarrolladores. El primero tiene la responsabilidad de transformar todo lo solicitado en requerimientos del Product Backlog, el segundo está encargado de satisfacer todas las necesidades para que el equipo sea exitoso y los desarrolladores son los encargados de realizar el software.
- **Product Backlog:** Es una lista creada por el Product Owner y por el equipo de desarrollo, en esta lista se plasman todos los requerimientos en orden descendente de qué tanto valor aporta al cliente o negocio. Esta lista es dinámica, es decir, se pueden agregar, modificar o eliminar requerimientos a lo largo del desarrollo.
- **Sprints:** Ciclos de trabajo en donde se desarrollan ciertas funcionalidades del producto, éstos tienen una duración de entre 1 a 4 semanas. Al momento de iniciar un sprint, no se puede agregar, quitar o modificar los requerimientos presentes en este. Una vez finalizado el sprint, en caso de no haber desarrollado todos los requerimientos, estos deberán ser notificados al cliente y se deberán de desarrollar durante el siguiente ciclo.
- **Sprint Planning:** Se realiza una vez al inicio de cada sprint, en donde se discuten las funcionalidades a realizar durante el sprint y se estimará el tiempo a emplear en el desarrollo de cada historia de usuario, a través de actividades como el *T-shirt Sizing* [8], *Planning Poker* [9], *Bucket System* [10] o *Dot Voting* [11]. La idea es analizar el Product Backlog y seleccionar las tareas en orden descendente, generando el Sprint Backlog.
- **Sprint Backlog:** El sprint backlog es un subconjunto de tareas obtenidas del product backlog. Estas tareas son ordenadas, priorizadas y estimadas durante el sprint planning y son propias de un sprint. Como se mencionó anteriormente en el punto de *Sprints*, este backlog no podrá ser modificado durante el transcurso del sprint.
- **Daily Standup:** Reunión diaria en un horario fijo con una duración aproximada de 15 minutos. La idea de esta sesión es que el equipo de desarrollo comunique las tareas realizadas, por realizar y los bloqueos o impedimentos que algún integrante haya enfrentado, con el propósito de que el Scrum Master pueda gestionar y buscar alternativas a dichos problemas.

- **Sprint Review:** Al finalizar cada sprint, se realiza esta sesión en donde participan los desarrolladores, Product Owner, Scrum Master, cliente(s) y stakeholder(s). Aquí es donde se presentan los avances del producto y se recibe retroalimentación, lo que permite la validación de las historias de usuario y la sugerencia de cambios, mejoras o nuevas funcionalidades, las cuales deberán ser planificadas y ordenadas dentro del tablero para el siguiente ciclo.
- **Sprint Retrospective:** Luego del Sprint Review, el equipo SCRUM se reúne en esta sesión para discutir qué ha ido bien y qué partes han tenido problemas respecto al proceso, con el fin de poder identificar las áreas a mejorar.

#### 2.4.2. Kanban

Kanban es un método visual de gestión en donde se visualiza el flujo de trabajo mediante un tablero que contiene tarjetas, las cuales representan tareas o elementos de trabajo y se mueven a través de columnas que indican diferentes etapas del proceso, como por ejemplo, “*Por hacer*”, “*En progreso*” y “*Hecho*”. Esto permite una gestión clara del trabajo en curso y de lo que está pendiente, facilitando la identificación de cuellos de botella. Además, el desarrollo de tareas se puede complementar con indicadores de importancia, como colores distintivos, columnas diferentes o indicadores que permitan a los desarrolladores identificar las tareas más importantes. Por último, Kanban también se centra en limitar el trabajo en curso para evitar la sobrecarga del equipo y en la mejora continua del proceso.

#### Prácticas comunes de Kanban

- **Visualizar el trabajo:** Una de las principales ventajas de Kanban es que se puede visualizar cómo avanza el trabajo a través de etapas. Por lo general, estas tareas se desplazan de izquierda a derecha, pasando por las columnas: “*Por hacer*”, “*En Progreso*” y “*Hecho*”. En ocasiones, es posible que este flujo de trabajo no sea suficiente y se requieran de columnas adicionales, como por ejemplo, “*En revisión*”, “*Rechazado*”, “*Bloqueado*”, entre otros.
- **Limitar el trabajo en curso:** Al centrarse en un principio de entrega temprana, las tareas deben moverse rápidamente de una columna a otra en lugar de estancarse en un estado de “*En Progreso*”. De esta manera, los desarrolladores se pueden concentrar en completar una tarea a la vez, mejorando la eficiencia y disminuyendo los tiempos de desarrollo.
- **Gestionar el flujo de trabajo:** El Project Manager o el encargado del proceso de desarrollo debe cerciorarse que el flujo de trabajo se mueva de manera rápida e identificar cualquier cuello de botella y riesgos.

- Implementar políticas de procesos explícitas: Establecer procedimientos, operaciones y reglas explícitas para todo el equipo de trabajo, con el fin de evitar problemas y malentendidos.
- Implementar ciclos de retroalimentación: Recopilar periódicamente comentarios de clientes y equipo, con el fin de identificar áreas a mejorar.
- Mejorar colaborando: Debido a que Kanban se trata sobre mejora continua, esta práctica promueve la experimentación y colaboración con otras metodologías o procesos.

### 2.4.3. Scrumban

Scrumban integra los mejores principios y herramientas de Scrum y Kanban, manteniendo la visualización del flujo de trabajo de Kanban e integrando la estructura de roles y eventos de Scrum. A diferencia de Scrum, Scrumban no está limitado por sprints fijos, lo que permite un flujo de trabajo más fluido y adaptable. Los equipos pueden utilizar tableros Kanban para visualizar el trabajo en curso y pueden introducir elementos de Scrum según las necesidades del proyecto y del equipo. Esta flexibilidad permite una adaptación más fácil a los cambios y una mejora continua del proceso.

#### Prácticas comunes de Scrumban

En esta sección se procederá a enumerar más no a definir las prácticas utilizadas por Scrumban, puesto que éstas ya fueron definidas en las secciones anteriores.

- Iteraciones
- Daily Standup.
- Sprint Retrospective.
- Visualizar el trabajo
- Limitar el trabajo en curso

### 2.4.4. Extreme Programming

Extreme Programming es un método de desarrollo de software riguroso y estricto que tiene como objetivo principal la creación de sistemas de alta calidad. Se basa en una estrecha interacción con los clientes, pruebas constantes y ciclos de desarrollo cortos para garantizar la entrega rápida y efectiva de productos que cumplan con las necesidades del cliente.

## Prácticas comunes de Extreme Programming

- Programación en pareja: La escritura de programación debe de ser siempre realizada en pareja, con la finalidad de optimizar la calidad de producto.
- Juego de planificación: Utilizada para guiar el trabajo, el fin de esta planificación serán los objetivos que se pretende alcanzar, los plazos previstos y los pasos a seguir.
- Pequeñas entregas: Se debe de realizar entregas pequeñas y periódicas para obtener retroalimentación durante el proceso. Estas entregas pueden ser enviadas entre el equipo de desarrollo o directamente hacia el cliente.
- Diseño simple: El diseño del producto se debe de mantener simple. Dentro de un estándar de XP, simplicidad se puede llegar a entender como que satisfaga los casos de prueba, no contenga código duplicado, refleje claramente la intención del programador y contenga la cantidad mínima de clases y métodos.
- Desarrollo guiado por pruebas (TDD): Es una forma de escribir código en donde se involucran pruebas unitarias. Inicialmente, se debe de contar solamente con casos de pruebas que fallan, ya que no existe código. Luego, se debe de escribir el código con la intención de que satisfaga las pruebas unitarias y finalmente, refactorizar de modo que el código siga los estándares establecidos.
- Equipo completo: Se establece que todos los involucrados en un proyecto trabajan como una única unidad integral. De esta manera, se fomenta la participación activa, la comunicación constante y la responsabilidad compartida entre los miembros del equipo.
- Integración continua: La idea es que el equipo de desarrollo esté siempre trabajando en la última versión del software. Debido a esto, es fundamental que cada par suba sus cambios continuamente para disminuir la probabilidad de surgir un problema de integración.
- Pruebas de usuario: El usuario o cliente debe de validar la historia de usuario a través de una prueba de aceptación.
- Metáfora: Facilitar la comunicación con el cliente, traduciendo particularidades del sistema y adaptándose a la realidad del usuario.
- Ritmo sostenible: Dentro de esta metodología, se busca la motivación y la armonía entre las partes implicadas en el desarrollo del proyecto. Debido a esto, los ciclos de trabajo no deben superar las 40 horas semanales.
- Estandarización del código: Los equipos deben de seguir un estándar, de modo que parezca que el código esté escrito por un solo desarrollador.
- Refactoring: Proceso que permite la mejora continua a través de la optimización del código, eliminando código duplicado y asegurar que el código sea coherente.

## CAPÍTULO 3

### PROPUESTA DE SOLUCIÓN

A continuación, se definirá la propuesta de solución, la cual buscará resolver las problemáticas planteadas en los capítulos anteriores. De esta manera, se definirán las prácticas de metodologías ágiles a utilizar, se establecerán las funcionalidades que debe realizar el software desarrollado, se realizará una breve introducción del negocio y sus actores y se definirán las tecnologías a utilizar en base a ventajas comparativas dependiendo de las limitaciones del negocio.

#### 3.1. Prácticas a utilizar

Se escogieron las siguientes prácticas, las cuales serán puestas a prueba y se determinará si tuvieron algún impacto a la hora de desarrollar el software.

##### 3.1.1. Kanban Board y Product Backlog

Se utilizará el tablero mencionado anteriormente de Kanban y se añadirá una columna con el Product Backlog de SCRUM. De esta manera, se tendrá en el extremo izquierdo del tablero el Product Backlog con las tareas a realizar ordenadas de manera descendente por orden de importancia. Seguidas de esta columna, se tendrán las columnas “*Por hacer*”, “*En progreso*” y “*Hecho*”. Para esta práctica, se utilizará Trello para tener un registro digital del tablero y poder administrar el proyecto de manera más eficiente.

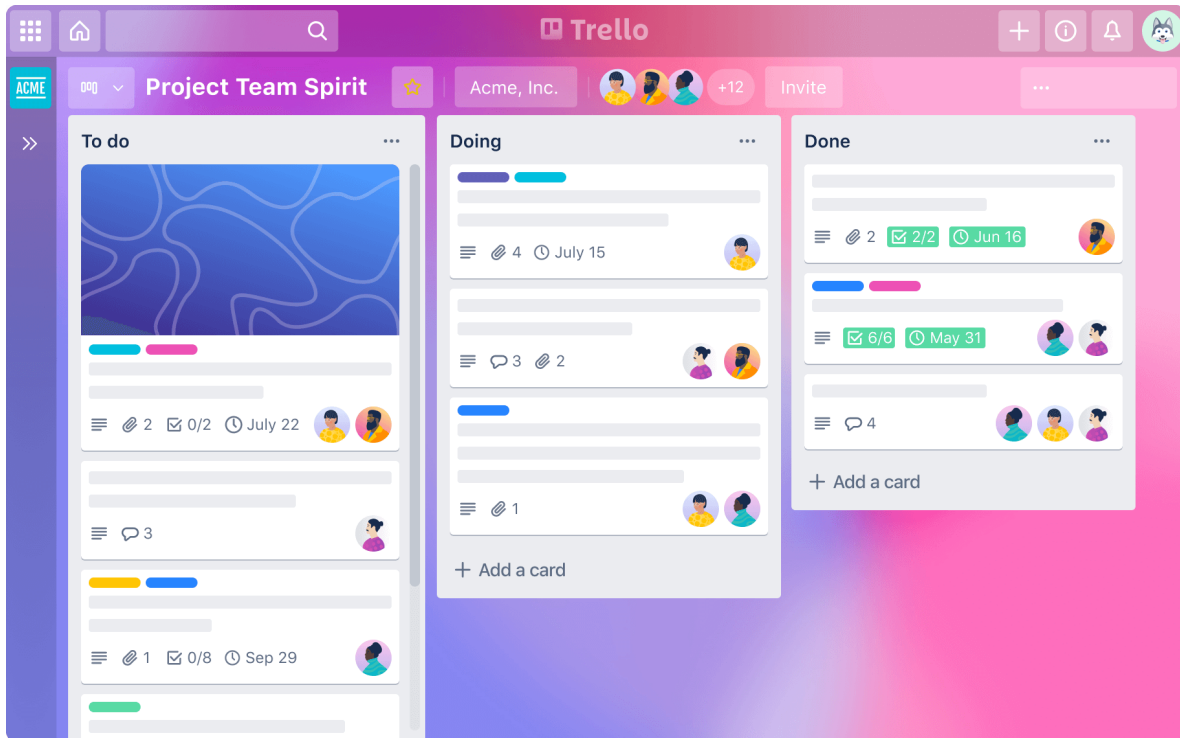


Figura 2: Ejemplo tablero Kanban.

Fuente: [12].

### 3.1.2. Limitar el trabajo

Se estableció una cantidad arbitraria de 2 tareas máximo en la columna de “En Progreso”.

### 3.1.3. Ritmo sostenible

Se estableció una cantidad arbitraria de 40 horas de trabajo máximo durante la semana.

### 3.1.4. Daily standup

Se realizó una modificación a la Daily Standup de SCRUM para que sea factible en un entorno de trabajo de una persona. De esta manera, en vez de realizar una reunión, se agendaron aproximadamente unos 5 minutos diarios para registrar dentro de una bitácora lo realizado el día anterior, si hubo problemas y qué se realizará durante el día.

Para esta práctica se utilizará *Journey*, una plataforma web que permite escribir bitácoras dentro de cada día, lo que permitirá registrar las actividades en la nube para un acceso más

fácil.



Figura 3: Vista actual de la sección de Cronología dentro de Journey.

Fuente: [13].

### 3.1.5. Iteraciones

Los períodos de trabajo se dividieron en iteraciones con una duración variable entre 2 a 3 semanas dependiendo de la cantidad de requisitos realizados. Se escogió este plazo debido a que se consideró que una semana de desarrollo es un período muy corto para mostrar avances relevantes, mientras que 4 semanas dilatarían mucho la iteración, teniendo en cuenta el tiempo que se tiene para el desarrollo del software.

### 3.1.6. Estandarización de código

A través de esta práctica, se permitirá tener un código más legible y facilitar la refactorización de código. Evaluando el uso de Java como lenguaje de programación, se utilizarán los estándares de código proporcionados por Oracle [14] y JetBrains [15], los cuales serán resumidos a continuación.

## Organización del código

La organización del código estará dividida por secciones, en donde cada sección estará separada por dos espacios en blanco. El orden a seguir es el siguiente:

1. **Paquetes e importes:** Se debe especificar el paquete con los importes necesarios para el correcto funcionamiento del archivo.
2. **Comentario de documentación de la clase/interfaz:** El cual se verá más a detalle en la siguiente sección.
3. **Declaración de clase o interfaz.**
4. **Comentario de implementación de la clase/interfaz:** El cual, al igual que el comentario de documentación, se verá más en detalle en la siguiente sección.
5. **Variables estáticas:** Las cuales deben seguir el orden de variables públicas, protegidas y finalmente privadas.
6. **Variables instanciadas:** Las cuales deben seguir el orden de variables públicas, protegidas y privadas.
7. **Constructores.**
8. **Métodos.**

Notar que no siempre estarán presentes todas las secciones dentro de un archivo de código, como por ejemplo, los comentarios de implementación o los constructores.

## Comentarios

Se utilizarán comentarios de documentación y comentarios de implementación. Los primeros tienen el propósito de añadir un contexto previo a una API, clases, métodos o módulos. Se deberán detallar los parámetros y valores de retorno en caso de haber, en caso contrario, se omitirán. De manera similar, los comentarios de implementación darán contexto al lector sobre alguna implementación en particular que requiera de alguna explicación para facilitar el entendimiento del código. En caso de que un comentario de implementación ocupe más de una línea, deberá empezar el comentario con los símbolos (/\*) y acabar con (\*), en donde cada línea entre medio de estos debe empezar con un asterisco (\*)

Ambos tipos de comentarios deben estar escritos en inglés y, en caso de haber múltiples comentarios de implementación dentro de un mismo bloque de código, estos deben estar en el mismo nivel de indentación.

---

```
// Example.java
```

```
/**
 * Example of documentation comment
 * The Example Class provides ...
 */
```

```
public class Example{
    /**
     * Example of documentation comment
     * @param string1 Some description ...
     * @param string2 Some description ...
     * @returns the Sum of two String numbers
     */
    private Integer addNumbers(String string1, String string2) {

        /**
         * Example of implementation comment
         * Converts string parameters to integers so they can be added.
         */
        Integer n1 = Integer.parseInt(string1); // Implementation comment
        Integer n2 = Integer.parseInt(string2); // Implementation comment

        return n1+n2;
    }
}
```

---

## Convención de nombres

Se utilizará una convención de nombre dependiendo de la parte del código, de esta manera:

- **Clases:** Las clases deben ser sustantivos y debe ser escrito en pascal case, es decir, el inicio de la palabra debe ser una mayúscula y, en caso de que se utilice una palabra compuesta, el inicio de cada palabra debe estar escrito en mayúscula.
- **Interfaces:** Las interfaces al igual que las clases, deben estar escritas con pascal case.
- **Métodos:** Los métodos deben ser verbos y deben de estar escritos con camel case, es decir, el inicio de la palabra debe estar en minúscula y, en caso de que se utilice una palabra compuesta, el inicio debe estar en minúscula y el inicio de las siguientes palabras debe estar en mayúscula.
- **Variables:** Los nombres de las variables deben ser fáciles de recordar y deben comunicar claramente su propósito a un lector casual. Además, deben escribirse en notación camel case. Por otra parte, se deben evitar nombres de variables de un solo carácter, excepto en casos de variables temporales, como aquellas utilizadas en los ciclos for.
- **Constantes:** Las constantes deben de estar escritas completamente en mayúsculas y, en caso de que se utilice una palabra compuesta, estas deben de estar separadas por un “\_”.

- **Archivos:** La convención de nombres variará dependiendo del tipo de extensión del archivo:
  - **.java:** Los archivos con extensión .java deben estar escritos en pascal case.
  - **.css:** Los archivos de estilos deben estar escritos en minúsculas y, en caso de que se utilice una palabra compuesta, cada palabra debe estar separada por un "-".
  - **.fxml:** Al igual que los archivos .css, los archivos que representan la interfaz de usuario deben de estar escritos en minúscula y se deben separar con un "-" las palabras compuestas.

A continuación se mostrará un ejemplo de una clase con todas las convenciones de nombres.

---

```
// Controller: ExampleClass.java
// UI: example-class.fxml
// Styles: example-class.css

public class ExampleClass extends Interface{

    private static final CONSTANT_NUMBER = 2;
    private Integer addNumbers(String firstString, String
        secondString) {

        Integer firstNumber = Integer.parseInt(firstString);
        Integer secondNumber = Integer.parseInt(secondString);

        return n1+n2+CONSTANT_NUMBER;
    }
}
```

---

## Composición sobre herencia

Se preferirá el uso de composición por sobre la herencia. La composición hace referencia a la creación de una clase más compleja a partir de la instanciación de otras clases, aislando de esta manera funcionalidades más específicas en otros componentes. Esto promueve un bajo acoplamiento, lo que facilita el mantenimiento y la modificación del código. Además, permite cambiar el comportamiento de un objeto en tiempo de ejecución al modificar sus componentes.

Por otra parte, la herencia permite a una clase heredar propiedades y métodos de otra clase, lo que facilita la reutilización de código y la extensibilidad. Sin embargo, puede resultar en un fuerte acoplamiento entre la clase base y las subclases, lo que puede dificultar el mantenimiento y la modificación del sistema a largo plazo.

## Pruebas unitarias

Antes de ejecutar el programa dentro de un ambiente comercial, se desarrollarán y ejecutarán pruebas unitarias para comprobar el funcionamiento correcto desde la unidad más pequeña de código. De esta manera, cada método deberá tener un test asociado, lo cual evitará errores que podrían presentarse en un contexto de producción.

## Declaraciones

Una declaración se refiere a cualquier instrucción que se escribe dentro del código y que el intérprete debe ejecutar. Cada declaración debe seguir convenciones determinadas dependiendo de su tipo:

- **Declaración simple:** Como una declaración o asignación de valor de una variable. Debe de haber como máximo una declaración por línea y no se recomienda el uso de comas “,” para agrupar múltiples declaraciones.
- **Declaraciones compuestas:** Una declaración compuesta es aquella declaración que contiene una lista de declaraciones encerrados entre llaves “{ }”. la primera llave “{” debe estar ubicado al final de la línea donde se inicia la declaración, mientras que la otra llave “}” debe iniciar una nueva línea y debe estar ubicado al inicio de la línea.
- **Declaraciones de retorno:** Este tipo de declaraciones entregan un valor al final de un método. Estas declaraciones no deben de utilizar paréntesis “( )” a excepción de que estos faciliten el entendimiento del valor de retorno.
- **Declaraciones condicionales:** Este tipo de declaraciones siempre debe de llevar llaves y, en caso de utilizar múltiples condicionales, estas declaraciones deben de empezar al finalizar el condicional anterior, como se muestra a continuación:

---

```
if (condition) {  
    statements;  
} else if (condition) {  
    statements;  
} else {  
    statements;  
}
```

---

- **Declaración de ciclos:** Los ciclos deberán de tener el siguiente formato:

---

```
for (initialization; condition; update) {  
    statements;  
}  
  
while (condition) {  
    statements;  
}
```

---

- **Declaración de switch:** Esta declaración deberá ser utilizada cuando se tengan múltiples condicionales que dependan de una sola variable y que entorpece la lectura del código. Esta práctica será automatizada a través del IDE desarrollado por JetBrains (IntelliJ), el cual detecta automáticamente cuando una declaración if puede refactorizarse por una declaración switch.
- **Declaración try-catch:** Se deberá emplear bloques try-catch para el manejo de excepciones, los cuales deberán tener el siguiente formato

---

```
try {  
    statements;  
} catch (ExceptionClass e) {  
    statements;  
}
```

---

En caso de que se requiera mostrar por consola la excepción, se preferirá el uso de la clase Logger de Java, ya que muestra la excepción de manera más legible y facilita el ruteo del error.

## Indentación

Se utilizarán 4 espacios en blanco como unidad de indentación, mientras que un tab debe representar 8 espacios en blanco.

Por otra parte, se preferirá que cada línea de código tenga como máximo 80 caracteres, rompiendo la declaración en múltiples líneas en caso de contener más caracteres, siguiendo las siguientes reglas:

- **R1:** Se deberá romper la declaración luego de una coma.

---

```
private void exampleMethod(longExpression1, longExpression2,  
                           longExpression3, longExpression4,  
                           longExpression5){... // Prefer
```

---

```
private void exampleMethod(longExpression1, longExpression2
                           , longExpression3, longExpression4
                           , longExpression5){... // Avoid
```

---

Notar que dentro del ejemplo no se sobrepasa el límite de 80 caracteres por línea al llegar a “*longExpression2*” y se realiza un corte, esto es debido a que en caso de realizar el corte donde es debido (“*longExpression3*”), la línea de código se desprende del contenedor del código dentro del documento. Debido a esto, para todos los ejemplos de indentación se realizarán los cortes cuando el texto se vaya a desbordar del contenedor de código.

- **R2:** Se deberá de romper antes de una operación.

```
int longName1 = longName2 * (longName3 + longName4 - longName5) + 4
                  * longname6; // Prefer

int longName1 = longName2 * (longName3 + longName4 - longName5) +
                  4 * longname6; // Avoid
```

---

- **R3:** Preferir cortes en niveles superiores a cortes en niveles inferiores, es decir, cuando se tenga una expresión anidada con múltiples niveles, como llamadas a métodos o estructuras con paréntesis, se debe de realizar un corte en expresiones superiores, es decir, que tengan un mayor grado de jerarquía a aquellas expresiones más internas.

```
int longName1 = longName2
                  * (longName3 + longName4 - longName5 + longName6)
                  * longname7; // Prefer

int longName1 = longName2 * (longName3 + longName4 - longName5 +
                             longName6) * longname7; // Avoid
```

---

- **R4:** Alinear la nueva línea con el comienzo de la expresión anterior, cerciorándose de que estén en el mismo nivel de indentación.

```
int longName1 = longName2
                  * (longName3 + longName4 - longName5 + longName6)
                  * longname7; // Prefer

int longName1 = longName2
    * (longName3 + longName4 - longName5 + longName6)
    * longname7; // Avoid
```

---

Para seguir estos estándares se utilizó la siguiente configuración dentro del IDE:

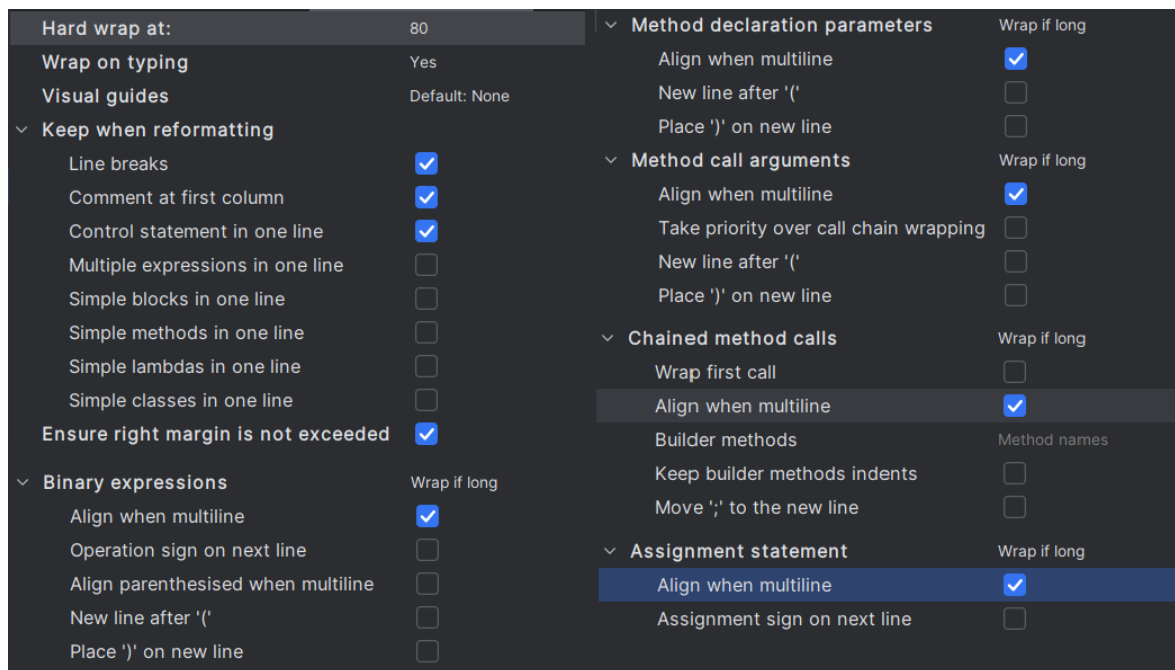


Figura 4: Configuración en la sección “Code Style/Wrapping and Braces” del IDE.  
Fuente: Elaboración propia.

En primera instancia se redujo el campo *Hard Wrap* de 120 a 80 y se marcó la opción de *Ensure right margin is not exceeded*, mientras que para cumplir R1 se seleccionaron las opciones *Wrap if long* para *Chained method calls*, *Method declaration parameters* y *Method call arguments*. Para R2, se seleccionaron las mismas opciones para *Binary expressions* y *Assignment statements*, mientras que para R4 se marcó la opción *Align when multiline* para todas estas secciones. Por su parte, no se encontró una manera de automatizar R3 de manera consistente, por lo que preferir cortes superiores se realizará de manera manual y la *preferencia* quedará a criterio del desarrollador.

### 3.2. Introducción del negocio

Se procederá a realizar una breve introducción del negocio al cual se le brindará el software, a modo de entender las características del mismo, tanto a nivel de infraestructura como de personal, que permitirá contextualizar y comprender aspectos relevantes para la elección del stack tecnológico.

Restaurante Hai San es un negocio vigente desde el año 2001, localizado en la región de Valparaíso y en la ciudad de San Antonio, el cual ofrece comida china cantonesa para servir y llevar y que consta de 2 pisos y una terraza, aunque actualmente solo están activos el primer piso y la terraza.

El negocio cuenta con un módem dentro del segundo piso que permite la conexión a internet, pero que, debido a la ubicación de este y la infraestructura del negocio, presenta una baja o nula señal dependiendo de la ubicación dentro del primer piso y, en algunos casos, la pérdida de conectividad momentánea.

El flujo de compra depende de qué tipo de servicio se requiera; para servir, llevar o despacho a domicilio. Dentro de la primera opción, el cliente ocupa una mesa a su elección para luego realizar el pedido a algún garzón. Luego, el garzón es el encargado de entregar la comanda al cajero y a los cocineros, siendo el cajero quien debe traspasar la comanda a otra hoja a modo de tener un registro más claro, mientras que los cocineros son los encargados de cocinar y emplatar los productos, los cuales son entregados al cliente a través del garzón. Finalmente, el cliente puede ordenar más productos o pedir la cuenta; en caso de ordenar más productos, se procede a replicar el proceso mencionado anteriormente y, en caso de pedir la cuenta, el cajero es el encargado de realizarla, mientras que el garzón es el responsable de entregar y realizar el cobro de la misma.

A continuación se presenta un diagrama de carril que refleja de manera visual los actores y sus respectivas acciones dentro de este proceso.

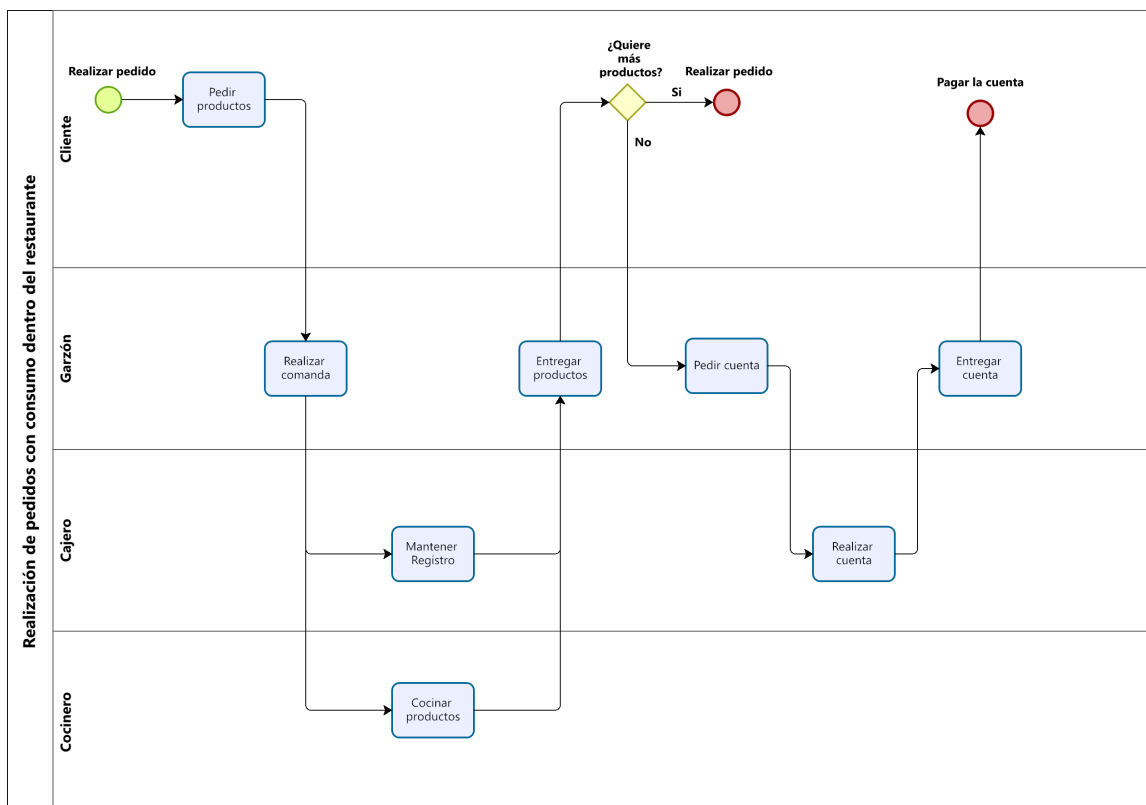


Figura 5: Visualización toma de pedidos con servicio dentro del restaurante.

Fuente: Elaboración propia.

En caso de realizar un pedido para llevar, los clientes deben acercarse a un apartado del restaurante, en donde el cajero es el encargado de la toma de pedidos para llevar. Luego de la realización de la comanda, se procede con el cobro del mismo y la entrega de la comanda a los cocineros, quienes cocinan los productos y entregan a los ayudantes de cocina para el empaquetamiento del mismo. Por último, el cajero es el encargado de entregar el pedido al cliente respectivo. A continuación se presenta su respectivo diagrama de carril.

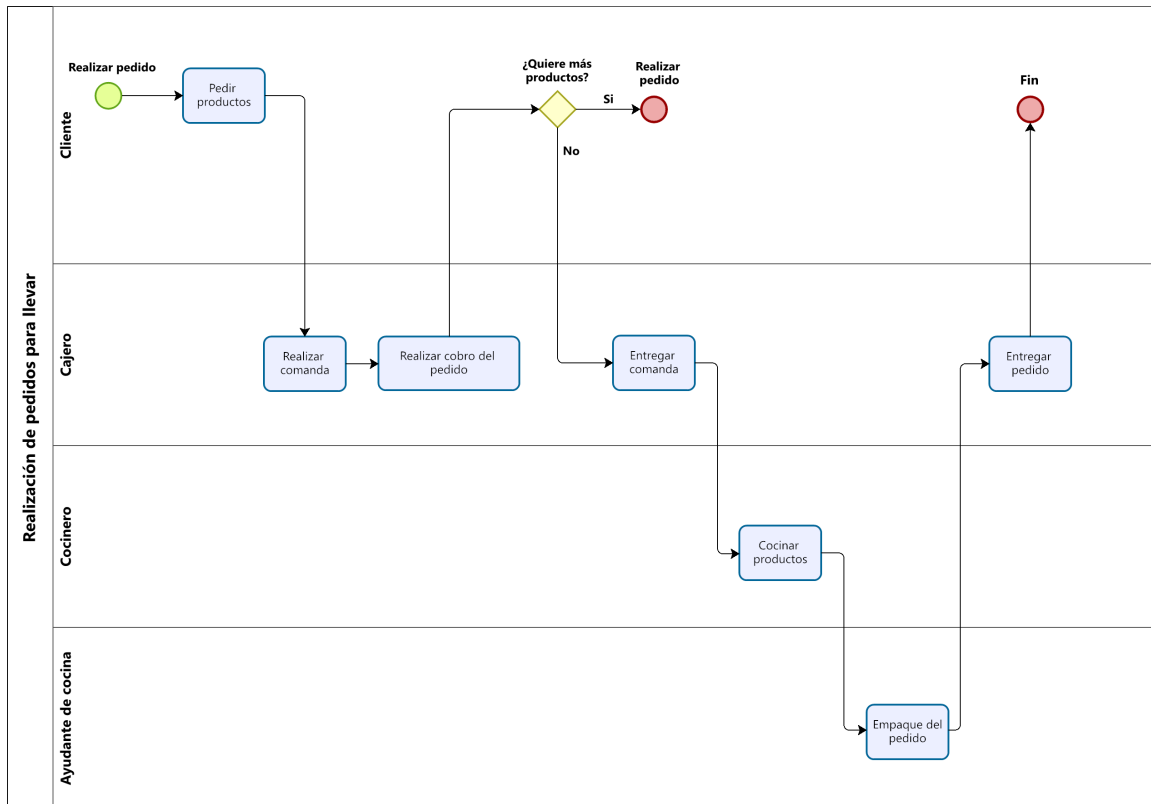


Figura 6: Diagrama de carril. Visualización toma de pedidos para llevar.  
Fuente: Elaboración propia.

En el tercer caso de que algún cliente quiera realizar algún pedido a domicilio, el consumidor debe realizar el pedido a través de la aplicación PedidosYa, la cual es la encargada de la recolección y entrega del pedido.

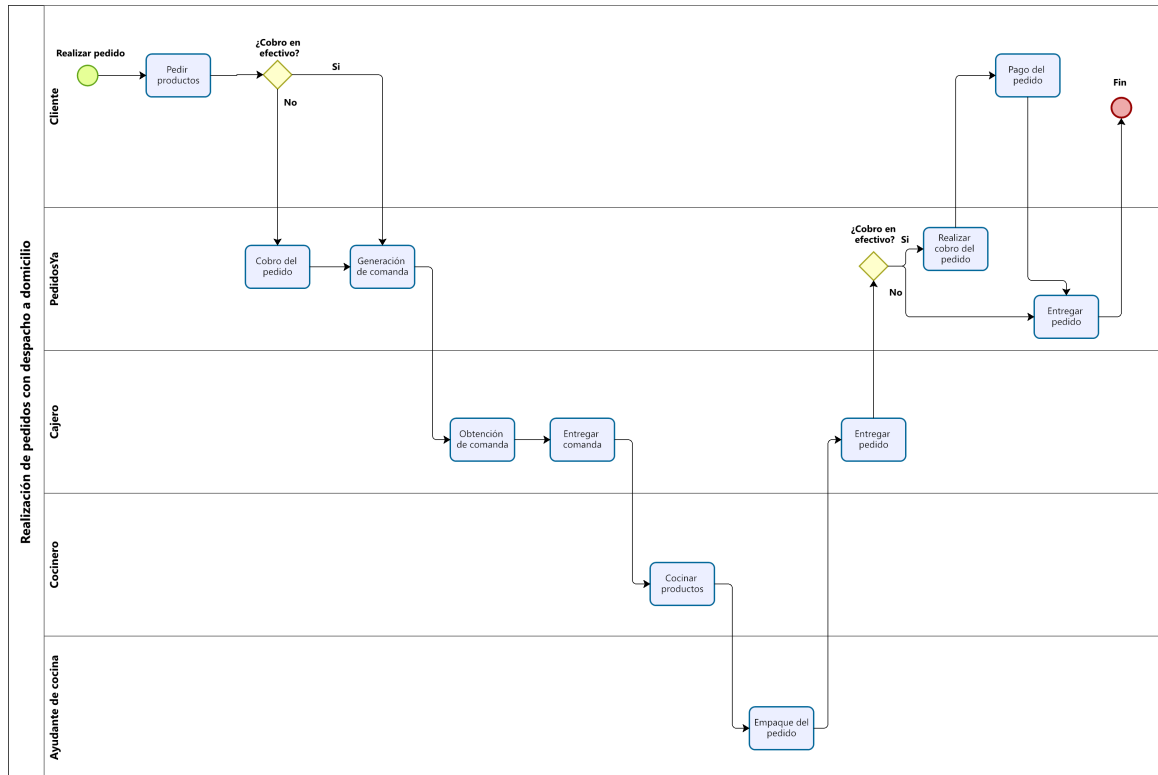


Figura 7: Diagrama de carril. Visualización toma de pedidos para envío a domicilio.  
Fuente: Elaboración propia.

Por otra parte, es recurrente que en algunos casos haya clientes que pidan productos que no están dentro de la carta, por ejemplo, el cambio de alguna materia prima en algún plato, cambio de platos dentro de algún menú, eliminación de alguna materia prima, etc. En estos casos, es posible que haya o no algún aumento del precio dependiendo del tipo de modificación que se realice.

Además, en algunas ocasiones se realizan descuentos dentro de algún pedido, ya sea por algún inconveniente que haya tenido algún cliente o simplemente se quiera realizar alguna rebaja a un cliente recurrente.

Al final del día y por temas de contabilidad, el negocio debe calcular las ventas totales, así como obtener la cantidad de propina total obtenida para distribuirla entre los garzones y revisar el inventario para el día siguiente.

Una vez presentado el negocio y, tomando en consideración la [Figura 5](#), [Figura 6](#) y [Figura 7](#), se procederá a identificar y resumir brevemente los actores que participan dentro de este:

### 3.2.1. Actores

#### Cliente

Es el actor principal del negocio, de esta entidad dependen los planes de negocios y acciones que tome el restaurante, ya sea agregar o eliminar algún producto del menú, realizar promociones, entre otros. Estos clientes se dividen en 3 tipos de clientes:

- Con consumo dentro del restaurante: Este actor por lo general será señalado dentro de este escrito como *mesa*.
- Para llevar.
- Con despacho a domicilio.

#### Garzón

Este actor se encarga de interactuar con los actores de la cocina y los clientes. Es el encargado de la entrega y cobro de los productos de mesas, así como también de estar al tanto de cualquier petición o inquietud que tengan. Dentro de algunos casos específicos, los garzones también toman pedidos de clientes para llevar. Actualmente se cuenta con 3 garzones.

#### Cajero o Administrador de negocio

Es el encargado de mantener registros de cada cliente con consumo dentro del restaurante, esto es, traspasar la comanda de cada mesa a una hoja indicando los productos consumidos, además de estar al tanto de comandas que reflejan productos adicionales solicitados por cada mesa. Además, el cajero debe tomar pedidos de clientes para llevar, estar atento a los pedidos a domicilio provenientes de PedidosYa, cobrar los pedidos para llevar y realizar la cuenta cuando las mesas lo soliciten. Por último, al finalizar el día, el cajero debe calcular las ganancias totales por temas de contabilidad. Se utilizarán los términos cajero y administrador de negocios indistintamente, ya que dentro de este contexto, los cajeros son los dueños del restaurante. Actualmente se cuenta con 2 cajeros.

#### Cocinero

Los cocineros realizan la preparación y emplatado de los productos. Estos actores no interactúan con ningún otro actor externo al ambiente de cocina. Actualmente se cuenta con 2 cocineros.

### Ayudante de cocina

Los ayudantes de cocina se encargan de realizar labores secundarias como el empaquetado de productos para llevar, lavado de loza, preparación de materias primas, etc. Al igual que los cocineros, estos actores no interactúan con otros actores externos al ambiente de cocina. Actualmente se cuenta con 3 ayudantes de cocina.

### PedidosYa

Es la única aplicación de despacho a domicilio que actualmente tiene cobertura dentro de la ciudad de San Antonio. Debido a esto, es un actor fundamental para los pedidos a domicilio.

A continuación se presenta un diagrama de contexto que sintetiza las interacciones entre los actores y el sistema a realizar.

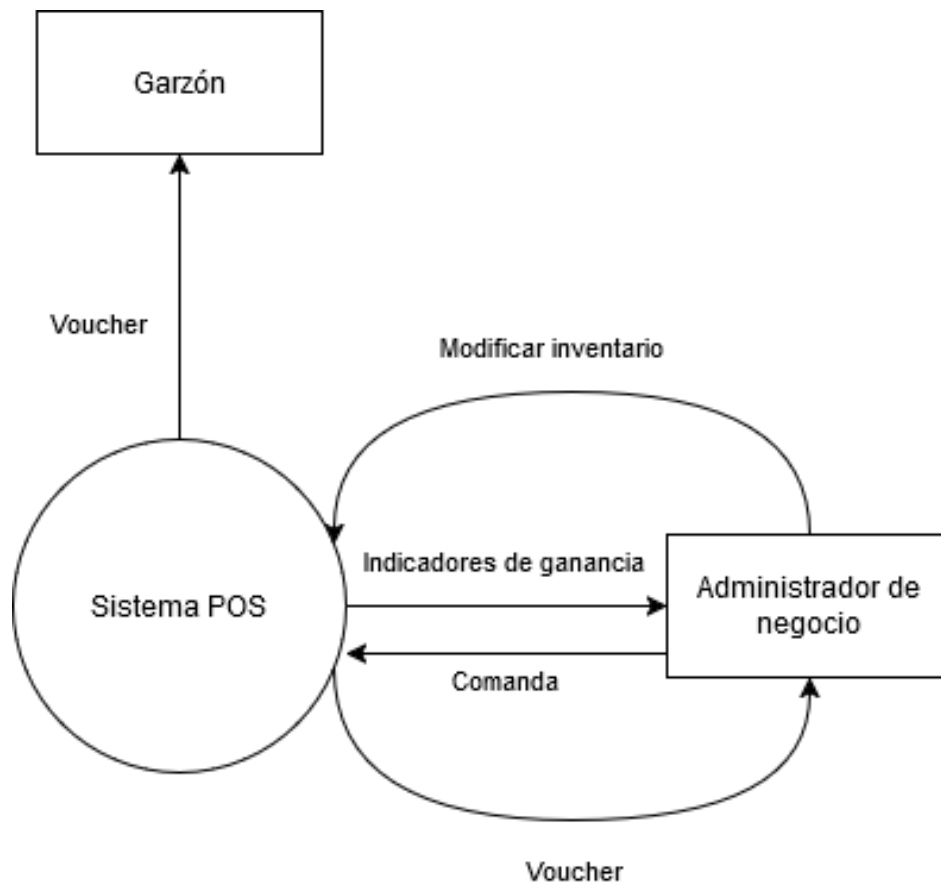


Figura 8: Diagrama de contexto del sistema POS.  
Fuente: Elaboración propia.

Cabe destacar que dentro de la **Figura 8** no se muestran todos los actores detallados con

anterioridad, esto debido a que forman parte del flujo del sistema, pero no interactúan de manera directa con éste.

### 3.3. Elección de tecnologías

Una vez establecidas las prácticas a utilizar y dado el contexto del negocio, se procederá a elegir el stack tecnológico a utilizar. Para esto, se compararán en cuanto a nivel de conocimiento, ventajas y desventajas que brinda cada opción y características propias del negocio que son relevantes para la elección del mismo.

#### 3.3.1. Frontend

Inicialmente se tuvo en consideración dos stack tecnológicos para la realización del proyecto, siendo el primero Javascript + React, mientras que el segundo es Java + JavaFX. En la [Tabla 3](#) se disponen las ventajas y desventajas de cada stack en base a las características de la empresa y el encargado de desarrollar el sistema.

|                                      | Javascript + React | Java + JavaFX  |
|--------------------------------------|--------------------|----------------|
| <b>Experiencia previa desarrollo</b> | Si                 | No             |
| <b>Tipo de aplicación</b>            | Web                | Escritorio     |
| <b>Acceso a impresora</b>            | Indirecto          | Directo        |
| <b>Necesidad de dominio</b>          | Si                 | No             |
| <b>Diseño responsivo</b>             | Fácil              | Difícil o Nulo |
| <b>Documentación</b>                 | Reciente           | Antigua        |
| <b>Conectividad a Internet</b>       | Necesario          | No necesario   |

Tabla 3: Ventajas y desventajas de cada lenguaje de programación.  
Fuente: Elaboración propia.

Como se observa en la [Tabla 3](#), no se dispone de experiencia previa en el desarrollo con Java y JavaFX, y la documentación disponible para estas tecnologías es considerablemente más antigua en comparación con la información más actual y accesible relacionada con JavaScript y React.

Por otra parte, al desplegar la aplicación en un entorno de producción con Javascript + React se necesitaría de un hosting, un dominio (en caso de requerirlo) y conectividad a internet. Se estudiaron algunos planes de hosting, omitiendo aquellos con suscripción gratuita como *Github Pages* o *Netlify* debido a que el software a desarrollar es de uso comercial y además porque sería una aplicación web no estática. Algunos precios de hosting son:

|                | Precio (CLP) por 1 año | Descuento por contratar múltiples años |
|----------------|------------------------|----------------------------------------|
| Bluehosting    | 38.900                 | Si                                     |
| HostGator      | 25.340                 | Si                                     |
| Hosting        | 99.900                 | Si                                     |
| PlanetaHosting | 49.900                 | Si                                     |

Tabla 4: Precios de hosting.  
Fuente: Elaboración propia.

Por otra parte, se estudiaron dominios en caso de que el usuario requiera algún dominio específico.

|             | Precio (CLP) por 1 año | Descuento por contratar múltiples años |
|-------------|------------------------|----------------------------------------|
| Nic Chile   | 9.990                  | No                                     |
| v2Networks  | 9.900                  | No                                     |
| HostingPlus | 15.000                 | No                                     |
| GoDaddy     | 9.999                  | Si                                     |

Tabla 5: Precios de dominios .cl.  
Fuente: Elaboración propia.

De las tablas [Tabla 5](#) y [Tabla 6](#) se desprende que, anualmente, el costo asociado al hosting y dominios puede variar entre \$25.340 en el caso de optar por la alternativa más económica sin dominio personalizado y \$114.900 al elegir la opción más costosa con dominio incluido. Por lo tanto, este es un aspecto relevante a considerar al evaluar la creación de una aplicación web.

Por último, en temas de creación de código, el manejo de la redimensión de la ventana es mucho más sencillo de manejar en Javascript + React que en Java + JavaFX, esto debido a la propia naturaleza de ambas tecnologías, ya que la primera está orientada a un entorno común multiplataforma de navegadores web, oscilando entre dimensiones de escritorio, tabletas y celulares, mientras que JavaFX está orientado a un entorno de aplicaciones de escritorio. De esta manera, si bien es factible redimensionar los elementos de un contenedor al agrandar la pestaña del software, puede llegar a ser un problema modificar la disposición de elementos al tener un tamaño de pestaña muy pequeño. Por otra parte, Java presenta un acceso mucho más directo, sencillo y nativo del propio lenguaje para la conexión e impresión de elementos, mientras que en React se deben instalar dependencias exteriores y la conexión tiende a ser más lenta y menos directa.

Tomando las consideraciones planteadas anteriormente y, al conversar con el cliente, se optó por utilizar Java + JavaFX para la realización del producto, esto debido a los problemas

de conectividad, el costo que supone la contratación de hosting y dominio y porque no se utilizará el software en dimensiones pequeñas tipo tablet o celular.

### 3.3.2. Base de datos

Debido a la simplicidad de la información de los datos, siendo mayoritariamente cadenas de texto, números enteros, fechas y objetos tipo *JSON*, se utilizará la base de datos relacional MySQL 8.0.36 y se tendrán las siguientes tablas:

- **Inventario:** Este modelo almacenará los productos del negocio, lo que permitirá dentro del programa crear algún producto, modificarlo y eliminarlo.
  - id (Integer) (PK)
  - name (Varchar256)
  - price (Integer)
- **Usuario:** Este modelo se encargará de proteger las vistas, restringiendo el acceso a usuarios que no tengan las credenciales de alguna cuenta.
  - id (Integer) (PK)
  - name (VARCHAR256)
  - password (VARCHAR256)
- **Orden:** Esto mantendrá un registro de las ventas para servir y llevar dentro del negocio. A través de esta tabla, se generarán índices de ganancias y datos de relevancia para la administración.
  - id (Integer) (PK)
  - isTable (Boolean)
  - products (JSON)
  - server (VARCHAR256)
  - startDateTime (DATETIME)
  - endDateTime (DATETIME)
  - total (Integer)

### 3.3.3. Entorno de desarrollo integrado (IDE)

Se utilizará IntelliJ IDEA Community Edition 2024.1.2 para la escritura de código. Se preferirá este IDE por sobre otros editores como VS Code con extensiones debido a que es un IDE

enfocado en el desarrollo de Java y que además permite importar las configuraciones del propio VS Code. Por otra parte, y como se observó dentro de la sección de estandarización de código, este IDE es fácilmente configurable para que se refactorice el código en base a ciertas reglas.

### **3.4. Especificación del software**

Dentro de esta sección se detallarán las funcionalidades del software, las cuales fueron atomizadas en historias de usuario con sus respectivos criterios de aceptación, estimación de prioridad y estimación de esfuerzo. A continuación, se presentan las consideraciones a tener en cuenta a la hora de realizar las estimaciones.

### **3.5. Estimación de esfuerzo**

Para la estimación de esfuerzo, se asignarán puntos de historia utilizando la sucesión de Fibonacci como referencia para los posibles valores que pueden llegar a tomar las historias de usuario. De esta manera, se utilizará la tabla obtenida de [16], la cual se muestra a continuación:

| Puntos de historia | Fibonnaci práctico                                                                                                                                                                                                                                                            |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 0                  | No se requiere esfuerzo o hay algún esfuerzo, pero no se entrega valor de negocio, por lo que no se acumulan puntos al realizar este trabajo. Un ejemplo es un cambio de comportamiento deseado derivado de la retrospectiva de Scrum.                                        |
| 1                  | Muy pequeño. Los desarrolladores sienten que entienden la mayoría de los requisitos y consideran que es relativamente fácil, probablemente el ítem más pequeño en el Sprint y muy probablemente completado en un día.                                                         |
| 2                  | Pequeño. Se requiere un poco de reflexión, esfuerzo o resolución de problemas, pero los desarrolladores han hecho esto muchas veces, por lo que tienen confianza en los requisitos. O bien, suena extra pequeño, pero se quieren asegurar un poco más.                        |
| 3                  | Promedio. Los desarrolladores han hecho esto muchas veces, saben lo que hay que hacer. Puede haber unos pocos pasos adicionales, pero eso es todo. Es improbable que necesiten investigar nada.                                                                               |
| 5                  | Grande. Este es un trabajo complejo, o los desarrolladores no lo hacen con mucha frecuencia. La mayoría de los desarrolladores necesitarán ayuda de alguien más en el equipo. Este es probablemente uno de los ítems más grandes que se pueden completar dentro de un Sprint. |
| 8                  | Extra grande. Esto tomará algo de tiempo e investigación, y probablemente más de un desarrollador para completarlo dentro de dos semanas,. Además, los desarrolladores necesitarán hacer varias suposiciones que incrementen el riesgo y podrían afectar la finalización.     |
| 13                 | ¡Advertencia! Este es un trabajo complejo con muchas incógnitas y requiere varias suposiciones para dimensionarlo. Es demasiado grande para completarse en un Sprint. En su lugar, divídelo en varios ítems que puedan completarse de forma independiente.                    |
| 21                 | ¡Peligro! Un “21” o un “34” refleja demasiada complejidad para realizarse dentro de un sprint. Necesitará ser más refinado. El gran tamaño también indica más riesgos, suposiciones y dependencias involucradas en la finalización de este ítem.                              |
| ?                  | ¡Peligro! Como desarrollador, no queremos hacer este trabajo como está actualmente escrito. Es muy complejo y no puede ser completado en el plazo de un sprint o iteración. Quizás los requisitos sean tan borrosos que estén llenos de peligros                              |

Tabla 6: Ejemplo de valores de historias de usuario.

Fuente: [16].

### 3.6. Estimación de prioridad

Para estimar la prioridad de una historia de usuario, se utilizará una variación de la matriz de Eisenhower [17]. Esta matriz se divide en cuatro cuadrantes según dos criterios: la urgencia, entendida como cuán pronto el usuario requiere la funcionalidad, y la importancia, definida como el grado en que su desarrollo impacta en el valor entregado por el software. En la **Figura 9** se presentan los cuadrantes y sus respectivos significados.

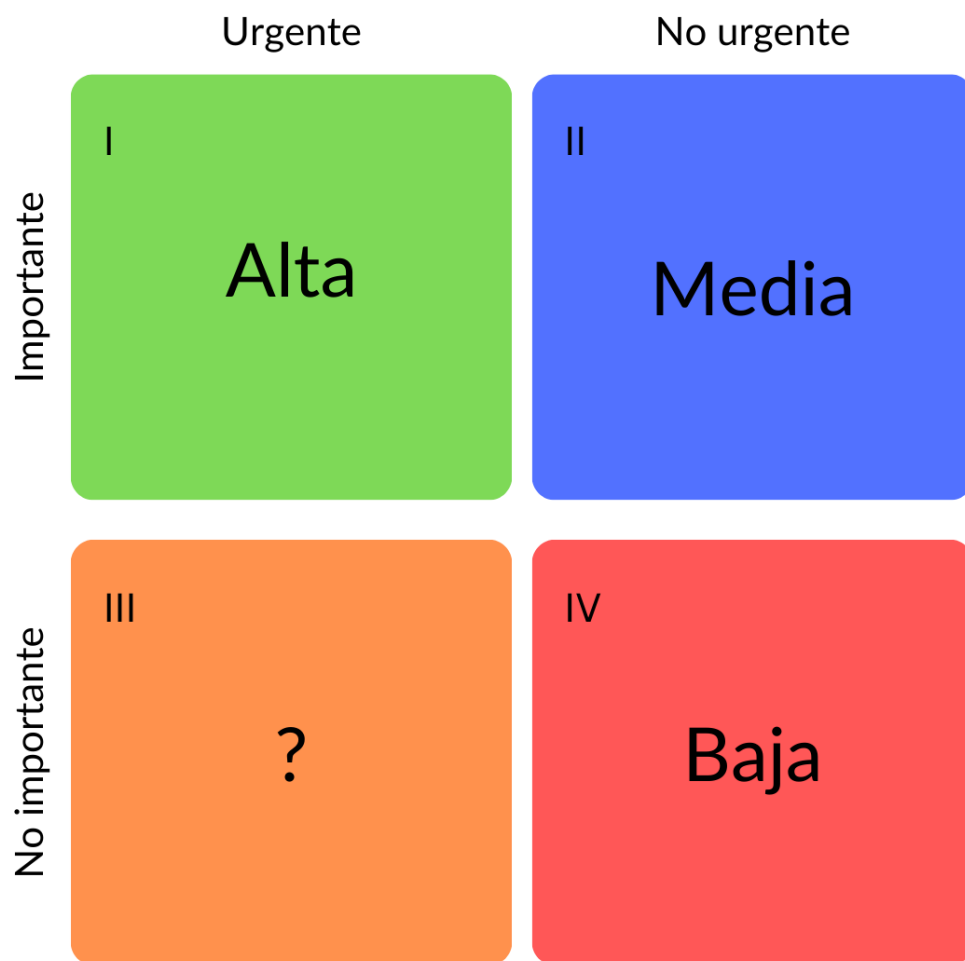


Figura 9: Variación de la matriz de Eisenhower.  
Fuente: Elaboración propia.

De esta manera, el primer cuadrante sugiere una tarea con prioridad alta, ya que el usuario la requiere lo antes posible y proporciona un alto valor al software. Por su parte, el segundo

cuadrante contiene tareas que, si bien aportan valor, no son necesarias de inmediato. Por otra parte, las tareas del tercer cuadrante se considerarán como tareas con prioridad baja; esto debido a que, si bien son consideradas urgentes, aportan un bajo valor al negocio, por lo que es posible que los clientes o partes interesadas en el software estén sobrestimando la urgencia de dicha tarea. Por último, las tareas del cuarto cuadrante son aquellas que se consideran de baja prioridad, ya que no aportan valor significativo y pueden aplazarse tanto como sea necesario sin afectar el desarrollo del producto.

### 3.7. Historias de usuario

#### 3.7.1. HU1

**Como** administrador de negocio, **quiero** ver indicadores de ganancia semanales y diarios, **para** poder visualizar el rendimiento de mi negocio.

#### Criterios de aceptación

- **CA1:** Dada la existencia de datos respecto al funcionamiento del negocio, cuando el administrador de negocio solicita ver estadísticas de rendimiento, entonces se mostrarán indicadores de ganancia de interés.

#### Puntos de historia: 5

Se considera como una historia grande, debido a que esta historia de usuario contempla la funcionalidad de una vista completa, además de implementar el uso de gráficos en JavaFX, lo que no se ha hecho con anterioridad.

#### Prioridad: media

Se considera una tarea con prioridad media, ya que la vista de métricas e indicadores de ganancia cuenta con una gran importancia, pero con una baja urgencia, debido a que no es fundamental para el funcionamiento del software.

#### 3.7.2. HU2

**Como** administrador de negocio, **quiero** registrar una venta para mesa, **para** mantener un registro de las mesas.

### Criterios de aceptación

- **CA1:** Dada una venta para mesa ya creada, cuando el administrador de negocio agrega un producto, entonces se incluye el producto al listado de productos del pedido y se actualiza el monto del mismo.
- **CA2:** Dada una venta para mesa ya creada, cuando el administrador de negocio elimina un producto, entonces se elimina el producto del listado de productos del pedido y se actualiza el monto del mismo.
- **CA3:** Dada la lista de productos de una venta para mesa, cuando el administrador de negocio selecciona un producto de la lista, entonces se mostrará el detalle del producto.

### Puntos de historia: 3

Se estima como una historia promedio. Es una historia de usuario que tiene similitud con la siguiente historia de usuario, pero con alguna diferencia en los datos que se pueden ingresar. Esta historia tiene 1 punto de historia más que la anteriormente mencionada debido a que esta cuenta con más funcionalidades.

### Prioridad: alta

Es una tarea con prioridad alta. Se establece con esta prioridad debido a que es parte fundamental de un sistema POS y, de no ser desarrollada inmediatamente, bloquea el desarrollo de otras tareas, como por ejemplo, la HU4, HU7 y HU8.

### 3.7.3. HU3

**Como** administrador de negocio, **quiero** registrar una venta para llevar, **para** mantener un registro de productos pedidos.

### Criterios de aceptación

- **CA1:** Dada una venta para llevar ya creada, cuando el administrador de negocio agrega un producto, entonces se incluye el producto al listado de productos del pedido y se actualiza el monto del mismo.
- **CA2:** Dada una venta para llevar ya creada, cuando el administrador de negocio elimina un producto, entonces se elimina el producto del listado de productos del pedido y se actualiza el monto del mismo.

- **CA3:** Dada la lista de productos de una venta para llevar, cuando el administrador de negocio selecciona un producto de la lista, entonces se mostrará el detalle de dicho producto.

#### **Puntos de historia: 2**

Se estima como una historia pequeña. Es una historia de usuario que tiene similitud con la historia de usuario de ventas para mesa, pero con alguna diferencia en los datos que se pueden ingresar.

#### **Prioridad: alta**

Es una tarea con prioridad alta. Al igual que la funcionalidad de ventas para mesa, es parte fundamental de un sistema POS y, de no ser desarrollada inmediatamente, bloquea el desarrollo de las tareas HU4, HU7 y HU8.

#### **3.7.4. HU4**

**Como** administrador de negocio, **quiero** poder generar los vouchers de venta, **para** entregárselos a los clientes.

#### **Criterios de aceptación**

- **CA1:** Dado el detalle de una venta para llevar o mesa, cuando el administrador de negocio seleccione la opción de generar voucher, entonces se generará un voucher con el detalle de venta.

#### **Puntos de historia: 5**

Se establece como una historia grande. Es una funcionalidad que no se ha realizado con anterioridad, por lo que podrían presentarse dificultades al momento de realizar la conexión con la impresora térmica y establecer el formato correcto de papel.

#### **Prioridad: media**

Es una funcionalidad que tiene una importancia alta, debido a que representa un gran impacto en el funcionamiento del software, pero no es necesario con inmediatez ni bloquea otras tareas, por lo que se considera con una prioridad media.

### 3.7.5. HU5

**Como** administrador de negocio, **quiero** ver todas las mesas, **para** poder acceder a los detalles de cada una.

#### Criterios de aceptación

- **CA1:** Dada la información de las mesas almacenadas, cuando el administrador de negocio solicite ver la información de las mesas, entonces se mostrará un listado de todas las mesas con su respectivo número de mesa y garzón asignado.
- **CA2:** Dado un listado de mesas, cuando el administrador de negocio solicite ver el detalle de una mesa, entonces se mostrará su detalle; productos pedidos, subtotal, descuento, monto total, garzón a cargo y número de mesa.

#### Puntos de historia: 2

Se considera como una historia pequeña. Es una funcionalidad que solo requiere la muestra de información con, posiblemente, estructuras ya utilizadas anteriormente.

#### Prioridad: alta

Es una tarea que se estima con una prioridad alta, debido a que proporciona la interfaz para poder acceder a los detalles de cada mesa, además de ser una tarea urgente debido a que bloquea tareas que tengan relación con pedidos para mesa.

### 3.7.6. HU6

**Como** administrador de negocio, **quiero** poder actualizar los productos de mi inventario, **para** tener un registro actualizado de mis productos.

#### Criterios de aceptación

- **CA1:** Dado un producto no almacenado, cuando el administrador de negocio ingrese todos los datos del producto correctamente, entonces se creará un nuevo producto.
- **CA2:** Dado un producto no almacenado, cuando el administrador de negocio ingrese algún dato de forma errónea, entonces se alertará el error y no se creará el producto.

- **CA3:** Dado un listado de productos almacenados, cuando el administrador de negocio seleccione algún producto, entonces se podrán modificar sus propiedades o eliminarlo.
- **CA4:** Dado un producto seleccionado, cuando el administrador de negocio modifique alguna propiedad y adicionalmente ingrese algún dato de forma errónea, entonces se alertará el error y no se actualizará la propiedad.

### **Puntos de historia: 5**

Se considera como una historia grande. Es una historia que contempla todas las funcionalidades de una sección. En esta historia se desarrollan múltiples vistas para poder agregar, modificar y eliminar productos.

### **Prioridad: alta**

Se estima una prioridad alta, debido a que esta es una historia de usuario con una gran importancia, debido a que la capacidad de almacenar, modificar y eliminar productos es fundamental para el funcionamiento de este tipo de sistemas, además de tener una gran urgencia debido a la dependencia de varias historias de usuario.

### **3.7.7. HU7**

**Como** administrador de negocio, **quiero** ingresar productos que no estén dentro de la carta **para** registrar excepciones en algunos pedidos.

### **Criterios de aceptación**

- **CA1:** Dado el ingreso de un producto que no esté en la carta, cuando el administrador de negocio ingresa al detalle de la venta, entonces se pedirá ingresar un “Producto fuera de menú” dentro del listado.
- **CA2:** Dado el ingreso de un producto que no esté en la carta, cuando el administrador de negocio ingresa de manera correcta todos los datos solicitados, entonces se añadirá el producto al listado de productos.
- **CA3:** Dado el ingreso de un producto que no esté en la carta, cuando el administrador de negocio ingresa algún dato de forma errónea, entonces se alertará el error y no se añadirá el producto.

### **Puntos de historia: 2**

Se considera una historia pequeña. Es una funcionalidad que podría considerarse como muy pequeña, pero por seguridad se asignaron 2 puntos de historia.

### **Prioridad: baja**

Se estima con una prioridad baja. Es una funcionalidad deseable, pero no fundamental para el funcionamiento del software. Por otra parte, es una tarea con urgencia baja, debido a que no bloquea ninguna otra tarea y el cliente no lo necesita con inmediatez.

### **3.7.8. HU8**

**Como** administrador de negocio, **quiero** aplicar descuento, **para** disminuir el precio de algún pedido.

### **Criterios de aceptación**

- **CA1:** Dado el detalle de una venta, cuando el administrador de negocio aplique un descuento con el formato correcto, entonces se disminuirá el monto total de la venta.
- **CA2:** Dado el detalle de una venta, cuando el administrador de negocio aplique un descuento con el formato incorrecto, entonces no se disminuirá el monto total de venta.

### **Puntos de historia: 1**

Se considera como una historia muy pequeña. No supone una dificultad en su desarrollo, pero aporta valor al negocio al ser desarrollado.

### **Prioridad: baja**

Se estima de baja prioridad. Es una tarea que, si bien aporta valor en su desarrollo, no es fundamental. Por otra parte, no bloquea otras tareas ni se necesita con inmediatez.

### **3.7.9. HU9**

**Como** administrador de negocio, **quiero** buscar un producto, **para** poder ver y modificar las propiedades de dicho producto.

### Criterios de aceptación

- **CA1:** Dado un listado de productos registrados, cuando el administrador de negocio ingrese texto dentro del campo de filtro, entonces se mostrarán las propiedades de los productos cuyos nombres contengan el texto ingresado.
- **CA1:** Dado un listado de productos registrados, cuando el administrador de negocio ingrese texto dentro del campo de filtro y adicionalmente no exista ningún nombre coincidente, entonces se mostrará un listado de productos vacío.

### Puntos de historia: 2

Es una historia que se considera pequeña, es posible que se utilicen contenedores y estructuras ya utilizadas anteriormente.

### Prioridad: baja

Es una historia con baja prioridad. Es una funcionalidad deseable debido a que mejora la experiencia de usuario, pero no es fundamental. Por otra parte, no bloquea otras tareas ni se necesita con inmediatez.

### 3.7.10. HU10

**Como** administrador de negocio, **quiero** almacenar las ventas concretadas, **para** mejorar la trazabilidad de órdenes.

### Criterios de aceptación

- **CA1:** Dada una venta para llevar o mesa ya creada, cuando el administrador de negocio imprima el detalle, entonces se almacenará la información relevante del pedido.
- **CA1:** Dada una venta para llevar o mesa ya creada, cuando el administrador de negocio la libere, entonces no se almacenará la información relevante del pedido.

### Puntos de historia: 2

Es una historia que se considera pequeña, ya que solamente se requiere almacenar la información relevante según el pedido del tipo.

**Prioridad: media**

Es una historia con prioridad media. Esto se debe a que es relevante para el cálculo de indicadores de ganancia.

### **3.8. Organización de iteraciones**

El desarrollo se dividió en cuatro iteraciones, cada una encargada de una vista del software o la creación de pruebas para la comprobación de su funcionamiento y las cuales tendrán una duración relativa, pero con un máximo de 3 semanas. En caso de que se excedan las 3 semanas y aún queden tareas por hacer, serán transferidas a la siguiente iteración como deuda técnica o, en caso de ser necesario, se alargará la iteración con su debida justificación.

A continuación se presentará el Product backlog con las tareas definidas y ordenadas de manera descendente por orden de relevancia. Para mayor claridad, se separó la columna "Backlog" en tres columnas, las cuales son ordenadas por importancia de izquierda a derecha y, dentro de su propia tarjeta, de arriba hacia abajo. Luego, se procederá a detallar todas las iteraciones, con los objetivos, observaciones, problemas o mejoras y el qué se hará en el próximo sprint.

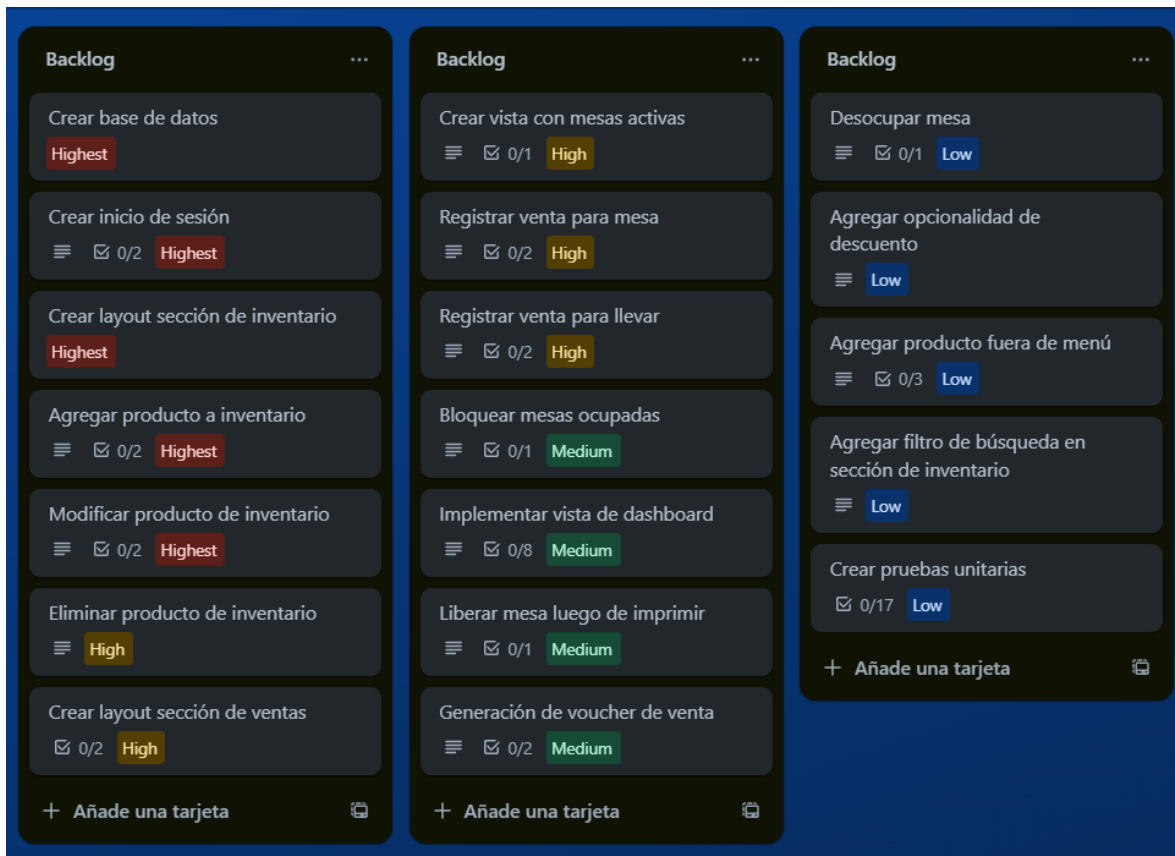


Figura 10: Vista del Product backlog.  
Fuente: Elaboración propia.

### 3.8.1. Primera iteración

#### Objetivos

Se empezó la primera iteración con la lectura y capacitación de las herramientas utilizadas por JavaFX. De esta manera, se dedicaron los primeros días del sprint al estudio de *FXML* y *SceneBuilder*, los cuales son utilizados por JavaFX para facilitar el desarrollo del área visual del software.

Una vez aprendido sobre las bases de estas herramientas, se procedió a levantar la base de datos local e implementar los modelos planteados en el apartado 3.3.2. Asimismo, se procedió con la realización de las funcionalidades de manejo de sesión y sección de inventario, las cuales son fundamentales para el funcionamiento del POS, pero que no presentaban dificultad de desarrollo, por lo que se condensaron durante esta primera iteración.

Este ciclo duró desde el 5 de junio hasta el 18 del mismo mes. En la Figura 11 se detallan las

tareas a realizar durante este ciclo.

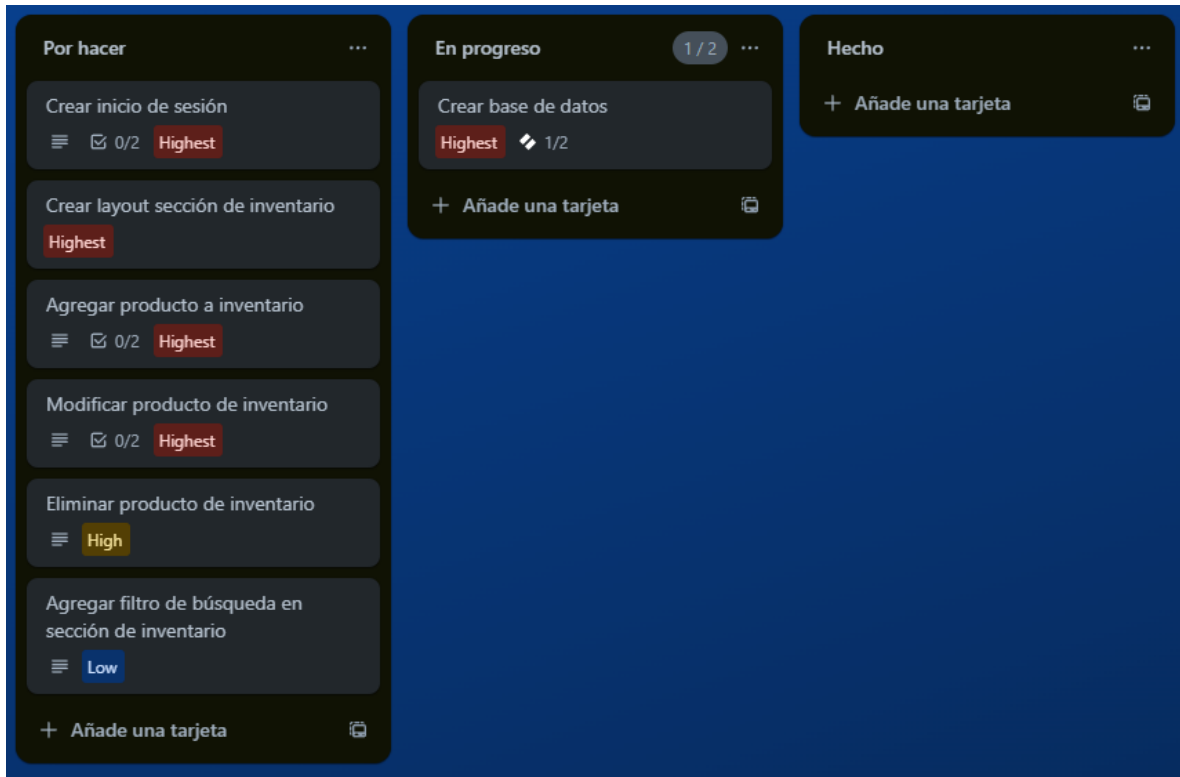


Figura 11: Vista del tablero durante la primera iteración. 05 de Junio.  
Fuente: Elaboración propia.

## Observaciones y problemas

Entre las dificultades presentadas durante esta iteración, se encuentra la falta de soporte de algunas propiedades propias de CSS dentro de JavaFX, como *flexbox*, *grid* o *media queries* que permiten un diseño responsivo y flexible. Esto se debe a que la disposición de elementos de manera responsiva es trabajada de manera directa a través de los contenedores de JavaFX, por lo que identificar los contenedores adecuados para cada situación fue fundamental y fue en donde se tuvieron problemas en un inicio.

## Qué se hará

Debido a que se completó el sprint en su totalidad y, una vez ya tenida la capacidad de almacenar, modificar y eliminar productos dentro del sistema, se procedió con la siguiente iteración sin deuda técnica.

### 3.8.2. Segunda iteración

#### Objetivos

Luego de conversar con el cliente, se estableció que lo ideal era que cada producto tuviese un tipo de producto asociado, por lo que se modificó el modelo relacionado al inventario y su sección correspondiente, agregando dicha propiedad al momento de agregar y modificar algún producto.

Una vez realizados los cambios solicitados por el cliente, se comenzó con el diseño general del layout de ventas, el cual contiene dos pestañas: “para servir” y “para llevar”. La diferencia entre ambas vistas radica en que las ventas para servir muestran las mesas que están siendo ocupadas, junto con una opción para agregar nuevas mesas, mientras que las ventas para llevar solamente muestran la opción de agregar una venta para llevar. Esta diferencia se debe al flujo de compra de ambos tipos de pedidos. Al momento de registrar una mesa, ésta puede estar varios minutos e incluso horas antes de realizar el pago del pedido, por lo que es necesario almacenar la información de múltiples mesas a la vez. Por otra parte, los pedidos para llevar son pagados, en su mayoría, justo después de haberse registrado el producto, por lo que no se consideró necesario registrar múltiples pedidos para llevar a la vez.

De esta manera, la vista de mesas cuenta con un contenedor flexible que permite mostrar todas las mesas del restaurante, el cual cuenta actualmente con 17 mesas, pero de las cuales algunas son divididas en mesas más pequeñas cuando no hay disponibilidad de mesas.

Una vez implementada la vista anterior, se procedió con la vista específica de una mesa u orden para llevar, teniendo la opción de agregar, modificar o eliminar un mesero, número de mesa y productos en caso de que sea una mesa, y solamente la opción de agregar, modificar o eliminar productos en caso de que sea una orden para llevar.

Para mantener el registro de los productos pedidos, se utilizó una tabla, en donde cada fila muestra la cantidad, el nombre del producto y el precio total. Al momento de seleccionar un producto, es posible modificarlo o eliminarlo.

Finalmente, se agregó la opción de agregar un producto que no esté dentro de la carta, para casos en donde algún cliente tenga alguna petición sobre algún producto que modifique su valor (por ejemplo, el agregado, reemplazo o eliminación de algún ingrediente).

Esta iteración comenzó el 18 de junio y finalizó el 19 de julio, superando el límite inicialmente establecido. La extensión se debió a que, al finalizar las tres primeras semanas del ciclo, quedaron en progreso la conexión con la impresora térmica y la generación del voucher de venta, las cuales son tareas fundamentales para el funcionamiento del sistema a desarrollar. Por ello, se optó por extender el ciclo una semana adicional para entregar esta vista completamente terminada y validar esta misma con el usuario. En la [Figura 12](#) se muestra el tablero al inicio de la iteración.

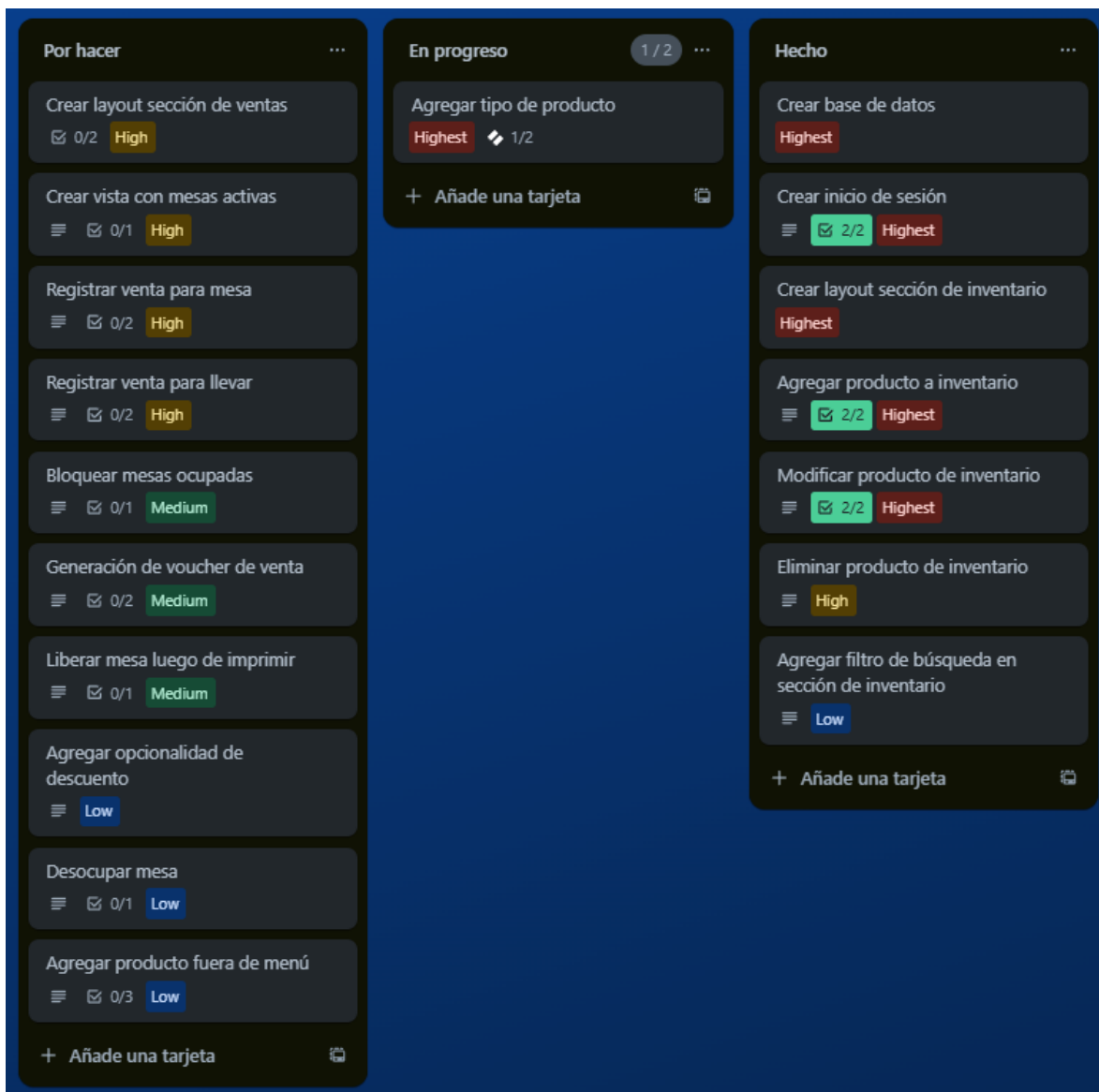


Figura 12: Vista del tablero durante la segunda iteración. 18 de Junio.

Fuente: Elaboración propia.

## Observaciones y problemas

Por otro lado, se identificaron limitaciones en ciertos contenedores que capturaban el input del usuario, impidiendo modificar ciertos comportamientos tanto en JavaFX como en SceneBuilder. Esto afectaba negativamente la experiencia de usuario, por lo que se optó por desarrollar un componente personalizado con funcionalidades similares.

Finalmente, se presentaron problemas relacionados con la generación del voucher, ya que el

texto impreso no respetaba el tamaño de hoja configurado, lo que ocasionaba que el texto se siguiera escribiendo al final de la línea en vez de realizar un salto de línea.

## Qué se hará

Debido a que no hay deuda técnica, se procedió con la siguiente funcionalidad. En este caso, solamente queda pendiente la funcionalidad de muestra y registro de indicadores de ganancia.

### 3.8.3. Tercera iteración

#### Objetivos

Una vez confirmadas las acciones de las funcionalidades desarrolladas con el cliente, este solicitó la capacidad de ver los pedidos cuyo voucher ya hubiera sido generado, con el fin de poder volver a editar una mesa ya cerrada, esto debido a que había situaciones en donde algunos clientes pedían productos adicionales luego de haber pedido la cuenta. Por otra parte, también era necesaria esta funcionalidad para borrar aquellos pedidos que fueron ingresados erróneamente.

Debido a esto, esta iteración comenzó realizando una sección de historial que permitiera ver los pedidos diarios, distinguidos por mesas y para llevar, cuyo voucher ya hubiera sido generado. Asimismo, dentro de esta sección se implementaron funcionalidades de filtro y ordenamiento, lo que ayudaría al cliente a encontrar los pedidos de interés con mayor facilidad.

Además de esta funcionalidad, el cliente solicitó la incorporación de un campo de observaciones, pensado para registrar información adicional que no esté directamente vinculada a los productos del pedido. A diferencia de los productos fuera de menú, estas observaciones permiten anotar detalles relevantes como el día y hora de retiro en pedidos para llevar, advertencias sobre alergias alimentarias o instrucciones específicas sobre el empaquetado, como separar ciertos productos en bolsas distintas.

Una vez desarrollados los cambios solicitados por el cliente, se prosiguió con la obtención y muestra de indicadores de ganancia. Para esto, en primera instancia se buscó información acerca de los gráficos proporcionados por JavaFX, entre los cuales se seleccionaron los gráficos circulares o “*pie charts*” para representar indicadores de ganancia con pocas categorías, como los productos más vendidos semanalmente y la distribución de mesas por garzón.

Para visualizar la distribución de mesas a lo largo del día, se optó por gráficos de líneas o “*line charts*”, mientras que los indicadores de ganancia restantes, como las ventas totales diarias y

semanales, las ventas para servir y para llevar (tanto diarias como semanales) y la retención promedio de mesas, se presentaron en formato de texto.

Esta iteración comenzó el 19 de julio con el desarrollo de la sección de dashboard y finalizó el 2 de agosto. A continuación se presenta el tablero al inicio del ciclo.

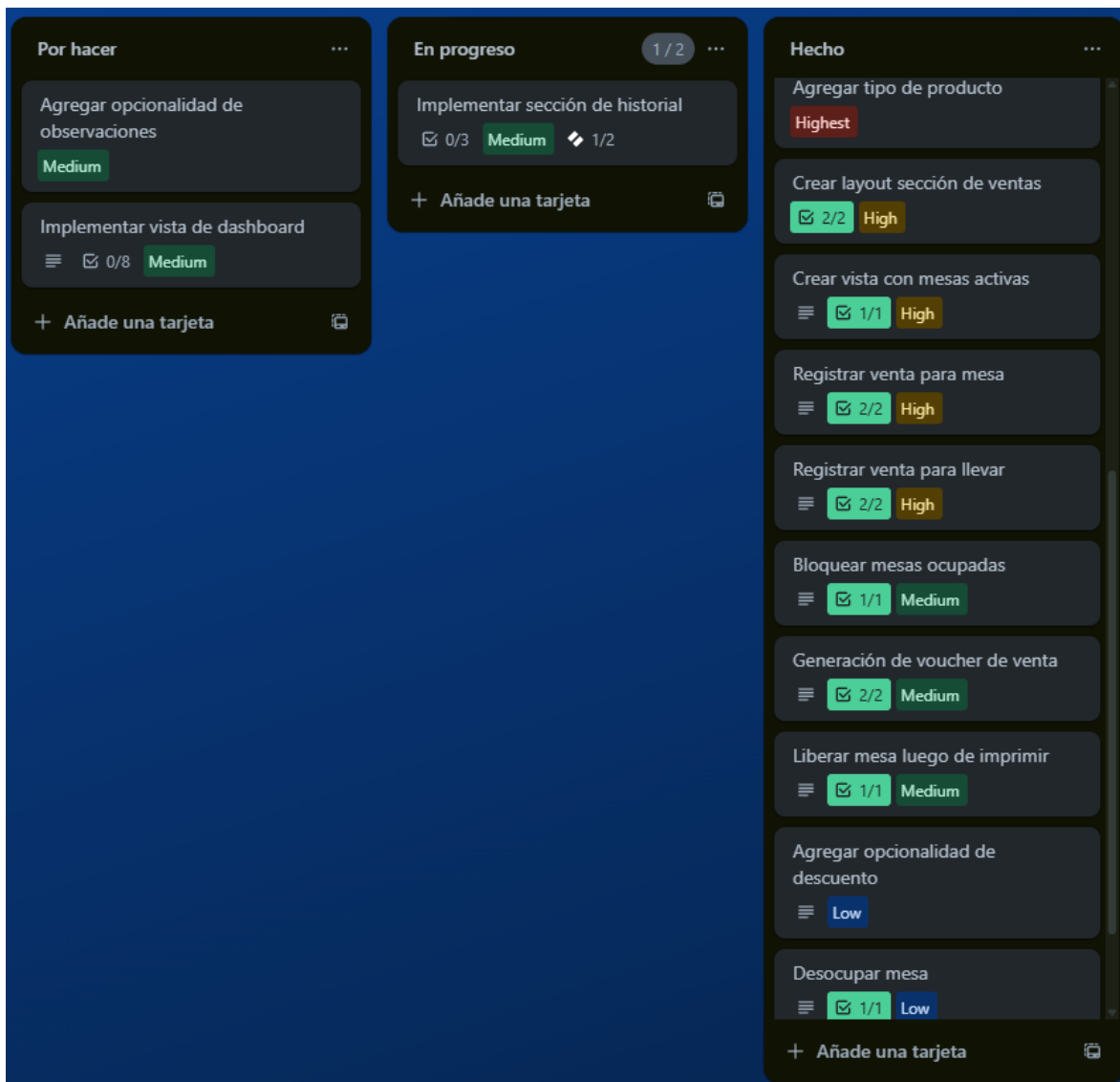


Figura 13: Vista del tablero durante la tercera iteración. 19 de Julio.  
Fuente: Elaboración propia.

### Observaciones y problemas

Durante el transcurso de este ciclo, se contó con problemas de animación de los KPIs, esto debido a que los indicadores de ganancia representados por gráficos, en específico los circulares, eran mostrados por pantalla ya rellenos al momento de entrar en la vista, mientras

que el comportamiento esperado era que comenzaran la animación de opacidad, tamaño y relleno una vez que el usuario entrara a la vista.

Por otra parte, al momento de ejecutar el programa a través de un archivo ejecutable, este tenía problemas al iniciar, cuyos errores eran debidos a que las variables de entorno y la clase principal no eran identificadas al momento de crear dicho programa.

### **Qué se hará**

Debido a que se finalizó la iteración sin deuda técnica, se procederá a avanzar a la siguiente iteración y, como no faltan funcionalidades por desarrollar, se procedió a realizar las pruebas unitarias del software.

#### **3.8.4. Cuarta iteración**

##### **Objetivos**

Al presentar los avances al cliente, este manifestó 3 requerimientos adicionales para mejorar el control y seguimiento de ventas.

En primer lugar, solicitó la distinción del voucher, ya que hasta el momento se estaba utilizando un voucher estandarizado tanto para ventas para servir como para llevar. De esta manera, dentro del voucher se modificaron los textos “Orden” por “Para llevar” o “Mesa”, según correspondiera, y se agregaron campos adicionales para diferenciar de mejor manera el tipo de venta.

En segundo lugar, solicitó la incorporación de un ID único en el voucher de venta. Cada pedido debía mostrar un identificador único autoincremental en el voucher, con numeración específica según el tipo de pedido (para llevar o mesa). Este id se reiniciaría semanalmente debido a la naturaleza del software.

Por último, pidió la implementación de un voucher de resumen general que incluya el detalle de todas las ventas del día. Para cada transacción, se deberá mostrar el id único de cada pedido, su tipo asociado (para llevar o mesa), el número de mesa cuando corresponda, y el monto total de cada pedido. El reporte debe mostrar subtotales independientes para ventas en mesa y ventas para llevar, incluyendo al final el total general de ventas del período.

Por otra parte, el cliente presentó algunas incomodidades a la hora de utilizar el software en etapa temprana, debido a que existían términos con los cuales no era familiar. A causa de estos problemas y, una vez que todas las secciones del software ya estuvieran realizadas, se procedió con la traducción completa de la aplicación y la implementación de un botón que

permitiera alternar los idiomas de español a chino simplificado y viceversa.

Luego, se comenzó con la creación de pruebas unitarias para cada método público de cada clase del software para cerciorarse de que cada método aislado estuviera funcionando correctamente. Para esto, se decidieron usar los siguientes frameworks:

- JUnit5: Permite probar cada unidad de código (método o clase) de manera independiente, además de estructurar y organizar pruebas eficientemente a través de la agrupación, etiquetado y ejecución de pruebas de forma selectiva.
- Hamcrest: Proporciona métodos para comparar tipos y objetos.
- TestFX: Para la simulación de interacción del usuario, con esto podemos simular acciones del usuario dentro de la interfaz, como por ejemplo, la modificación de algún producto, el cambio de vista, ingresar valores dentro de inputs, etc. Y comprobar que dichos valores sean correctos a través de los frameworks mencionados en el punto anterior.

De esta manera, para cada método, se verificó que retornara el valor correcto o realizara la acción esperada al preparar una situación válida y, en caso de que se entregara un valor o situación inválida, se comprobó que el software retornara un error o alguna alerta.

Esta iteración comenzó el 2 de agosto y finalizó el 17 del mismo mes. A continuación, se presenta el tablero al principio del presente ciclo:

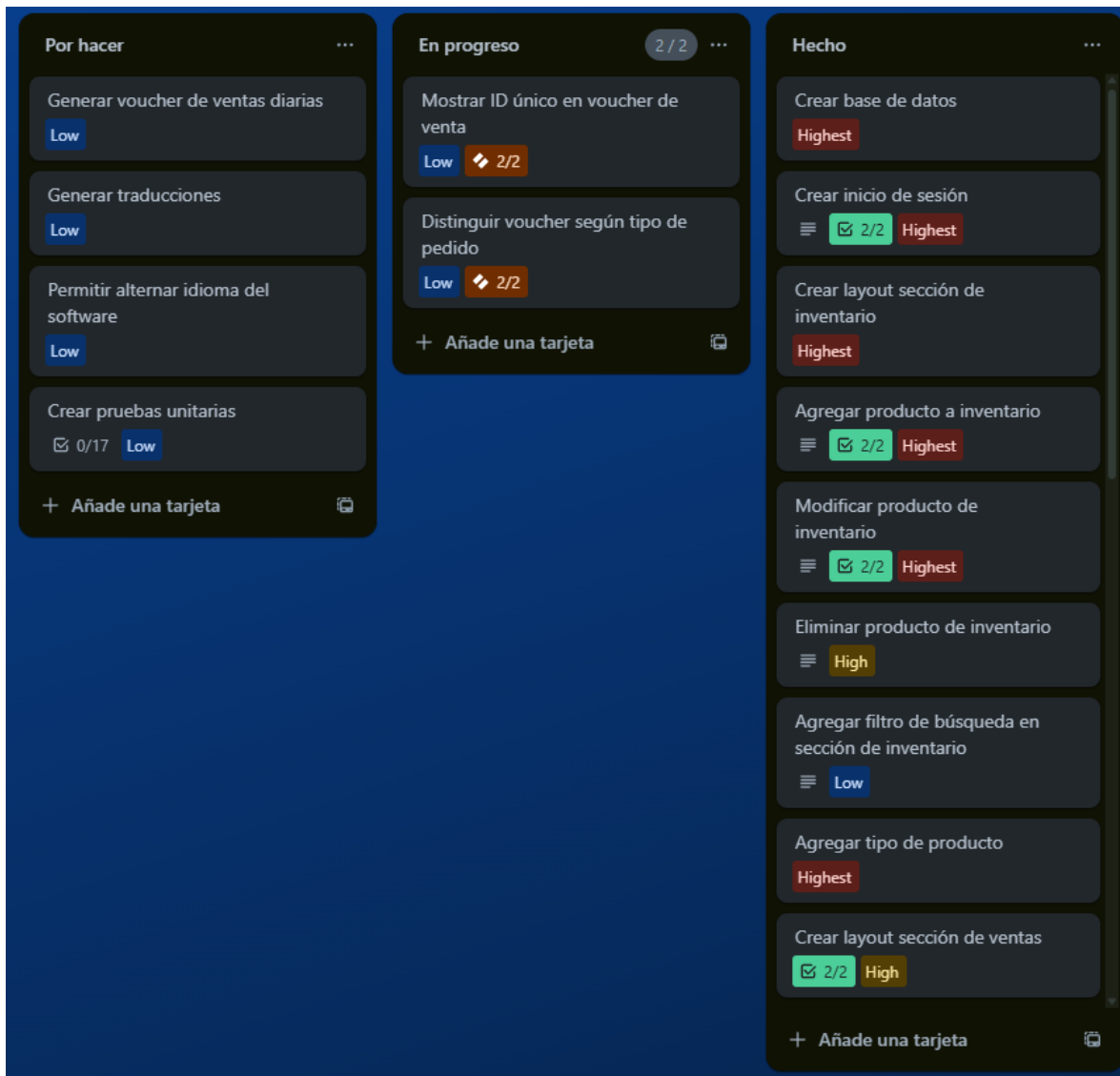


Figura 14: Vista del tablero durante la cuarta iteración. 02 de Agosto.  
Fuente: Elaboración propia.

## Observaciones y dificultades

Surgieron problemas mínimos al momento de realizar este sprint relacionados con la carga tardía de los contenedores de JavaFX sobre las comprobaciones realizadas por los tests. De esta manera, fue frecuente encontrar errores al momento de verificar algún tipo u objeto y que el valor de este fuera nulo o que el simulador no pudiera encontrar el contenedor con el cual interactuar. Esto se debía a que las comprobaciones se realizaban antes de que la clase se cargara por completo, por lo que se tuvo que utilizar métodos auxiliares que priorizaran la carga de la clase antes de realizar comprobaciones.

### **Qué se hará**

Finalizado el desarrollo y prueba del software, se procederá a comprobar su aplicación dentro de un entorno productivo.

## CAPÍTULO 4

### VALIDACIÓN DE LA SOLUCIÓN

En esta sección, se procederá a establecer los métodos por los cuales se validará la propuesta de solución definida en el capítulo anterior. En donde, en primer lugar y, a modo de contexto, se mostrarán las vistas desarrolladas del software.

#### 4.1. Vistas del software

##### 4.1.1. Inicio de sesión

En primera instancia, se le presentará la siguiente vista al usuario, quien podrá autenticarse con las credenciales de acceso previamente entregadas y, una vez autenticado, acceder a las demás vistas.

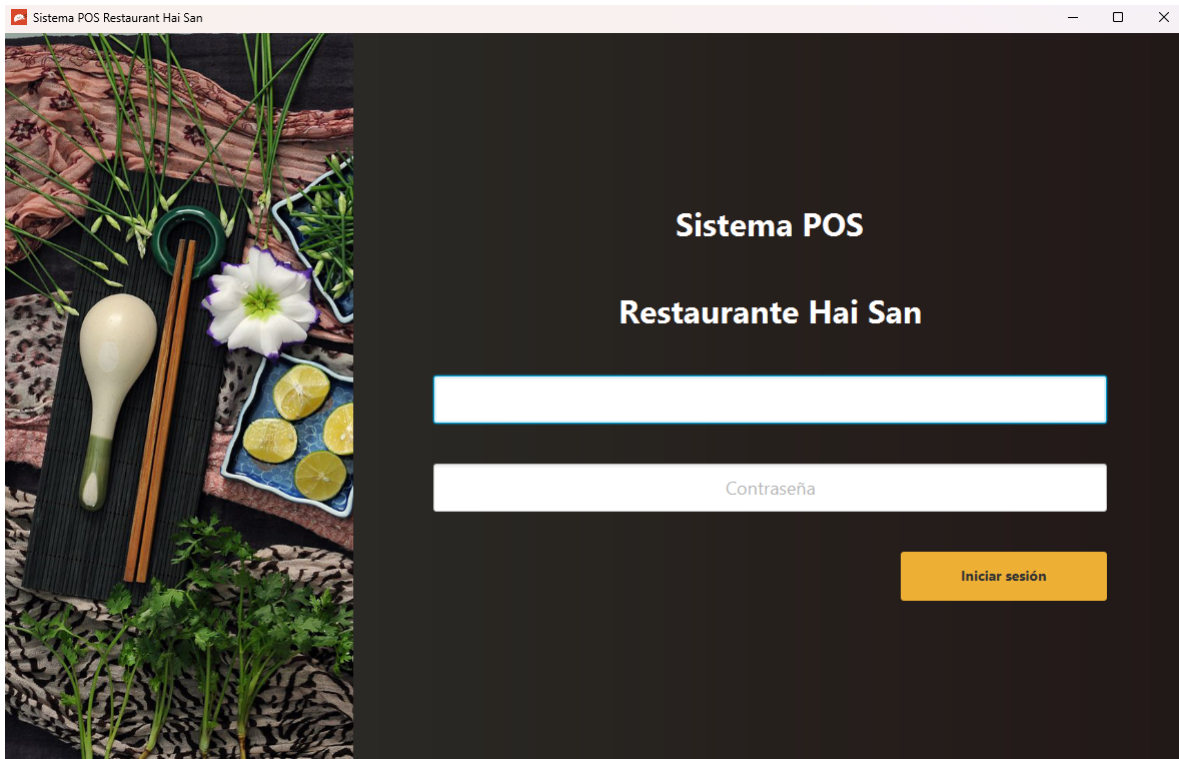


Figura 15: Vista del software, sección de login.  
Fuente: Elaboración propia.

#### 4.1.2. Dashboard

En esta vista, el usuario puede consultar los indicadores de ganancia solicitados, los cuales se actualizan automáticamente cada vez que una mesa solicita la cuenta. Para asegurar que estos indicadores representen únicamente la semana actual, al acceder a la vista se eliminan los registros cuya fecha sea anterior al lunes de la semana en curso. De este modo, los KPIs se reinician cada siete días. Este mecanismo se implementa así debido a que el estado del computador es dinámico; en ocasiones puede estar apagado, bloqueado o el sistema POS cerrado, lo que haría poco confiable la ejecución de una tarea programada a una hora fija.

Los indicadores de ganancia a mostrar son los siguientes:

- Ganancias diarias (en CLP): Separados en ventas para llevar y para mesas.
- Ganancias semanales (en CLP): Separados en ventas para llevar y para mesas.
- Pedidos diarios: Separados en ventas para llevar y para mesas.
- Pedidos semanales: Separados en ventas para llevar y para mesas.
- Distribución de mesas diaria por hora: La cual despliega la cantidad de mesas por hora en tiempo real.
- Cantidad de mesas por garzón: La cual entrega información sobre la cantidad de mesas asignadas semanalmente por garzón.
- Distribución de productos más vendidos: Para analizar los productos más vendidos semanalmente.
- Retención promedio de mesas: Para ver el tiempo promedio en el que una mesa llega al restaurante hasta su ida.

En la **Figura 16** se muestra la vista con algunos indicadores de ganancia.

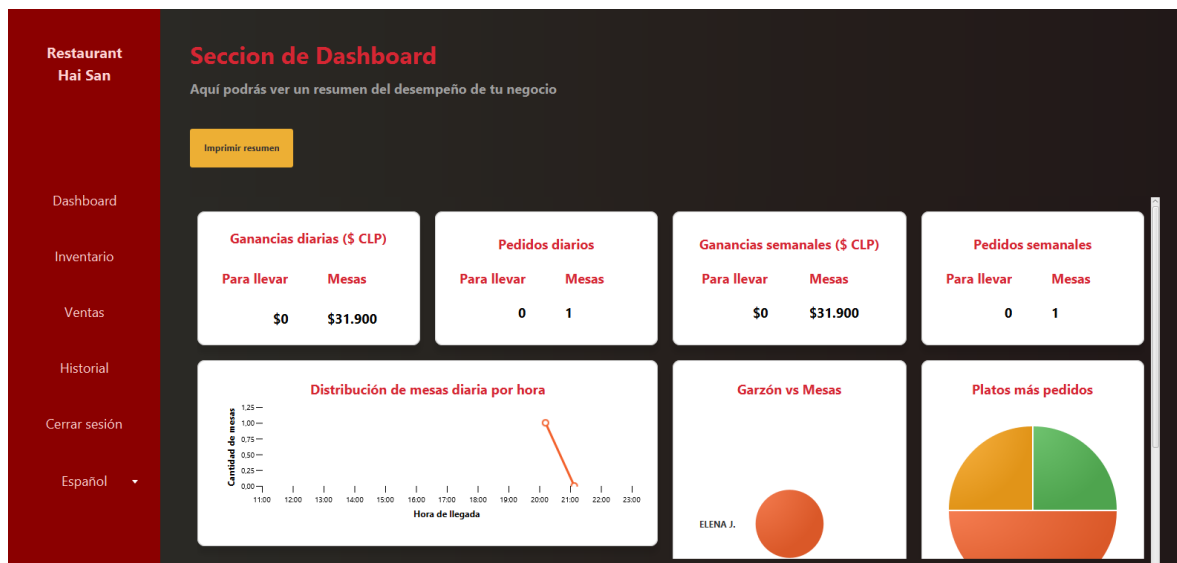


Figura 16: Vista del software, sección de dashboard.

Fuente: Elaboración propia.

#### 4.1.3. Ventas

El presente módulo concentra la parte más importante del negocio: el ingreso, retención y la salida de información (como un recibo de venta o voucher). Por lo anterior, dentro de las principales funcionalidades provistas al usuario es posible encontrar los botones “Crear mesas” o “Crear pedidos para llevar”, pudiendo agregar los productos pedidos por los clientes, realizar descuentos, agregar alguna observación, realizar algún cambio dentro de un producto pedido, obtener el recibo de venta acorde a la mesa o desocuparla, en caso de que se requiera liberar la mesa sin imprimir un voucher ni agregar la información a los indicadores de ganancia, funcionalidad que podría resultar útil para situaciones en donde se puede llegar a ingresar una mesa no existente por error.

Es importante validar la unicidad de pedidos del tipo mesa, puesto que no tendría sentido tener dos mesas con el mismo número de mesa. Debido a esto, cada vez que el administrador de negocio ingresa o modifica un número de mesa a uno existente, entonces se alertará que la acción no es permitida.

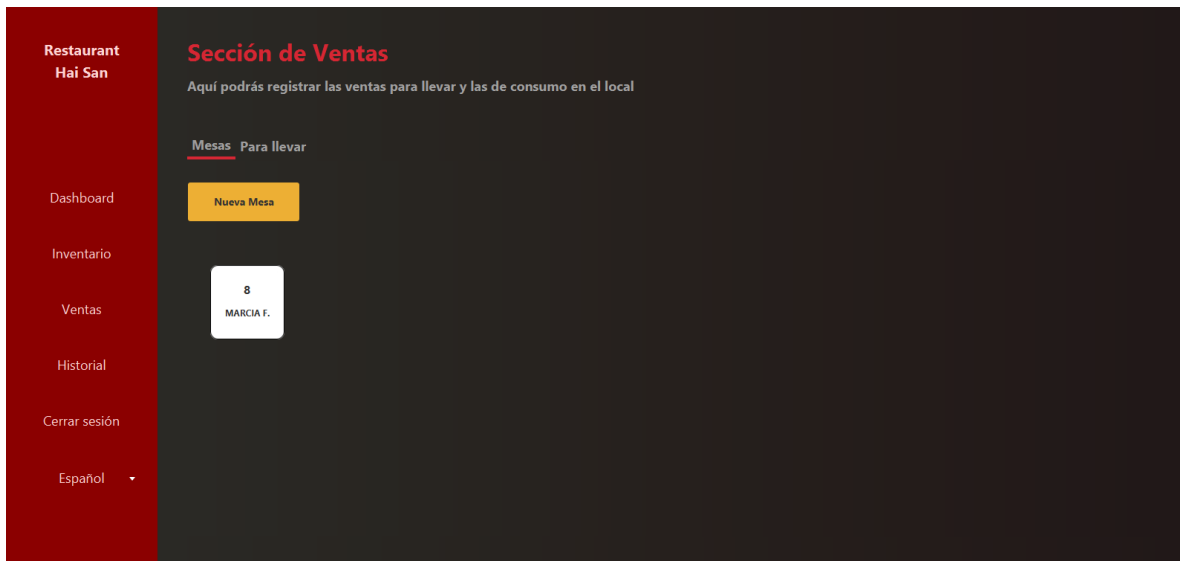


Figura 17: Vista del software, sección de ventas para llevar y mesas.  
Fuente: Elaboración propia.

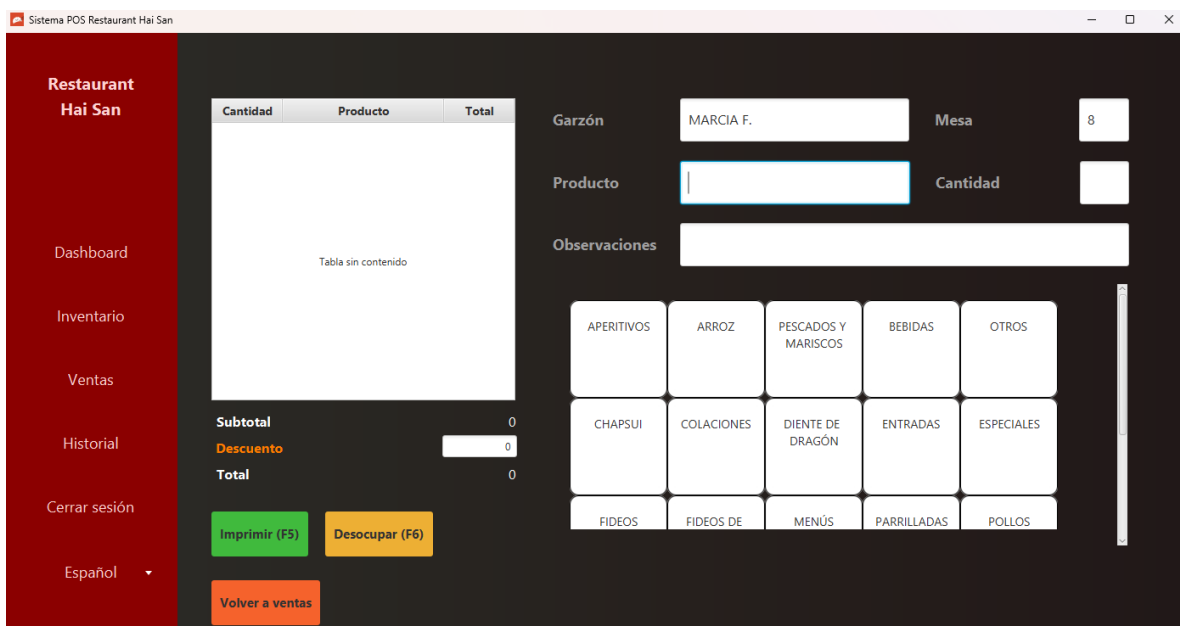


Figura 18: Vista del software, sección de una venta para mesa en específico.  
Fuente: Elaboración propia.

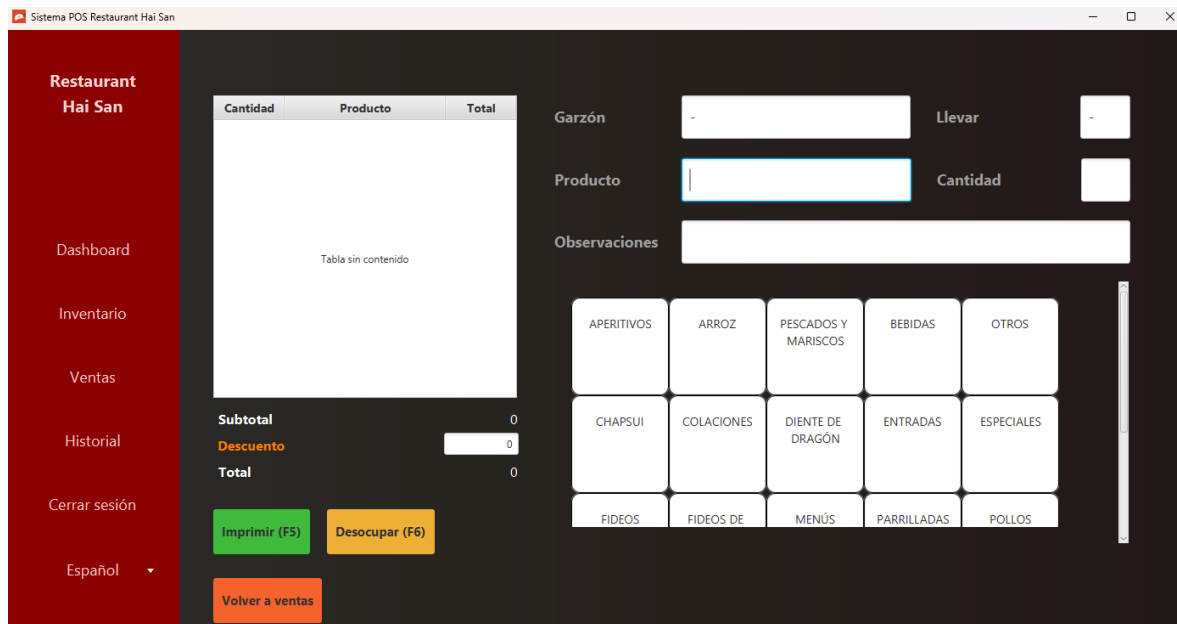


Figura 19: Vista del software, sección de una venta para llevar en específico.  
Fuente: Elaboración propia.

#### 4.1.4. Inventario

Esta sección es la encargada de proveer al usuario la capacidad de ingresar los productos que el negocio está vendiendo actualmente. De esta manera, en esta vista se puede ingresar un producto siempre y cuando se sigan las siguientes reglas:

- El campo de nombre de producto no está vacío y no tiene un nombre repetido (case-insensitive)
- El campo de precio de producto no está vacío
- El campo de tipo de producto no está vacío.

En caso de que alguna de estas reglas no se cumpla, se alertará al usuario por medio de un mensaje, indicando el campo y su respectivo motivo.

Por otra parte, dentro de la misma sección, es posible modificar y eliminar un producto; opciones que son mostradas en la misma tabla de productos. En caso de la modificación de un producto, el usuario es capaz de modificar el nombre, precio o tipo a través de un doble clic, mientras que para eliminar un producto basta con que el usuario presione el ícono asociado a la fila del producto.

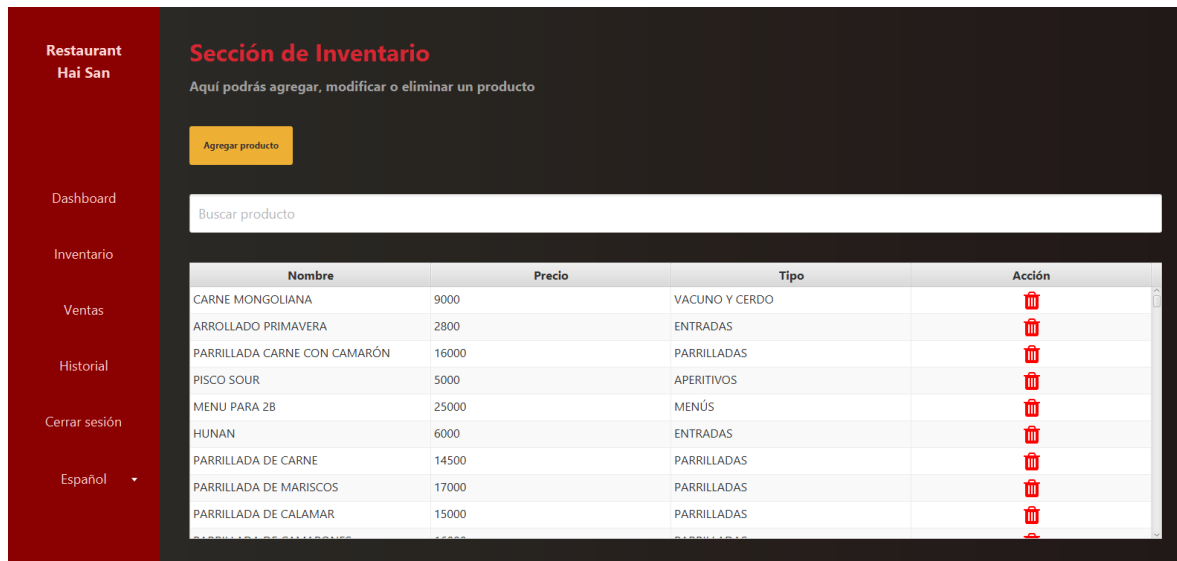


Figura 20: Vista del software, sección de inventario.

Fuente: Elaboración propia.

#### 4.1.5. Historial

Dentro de esta sección se le presentan al usuario las transacciones diarias realizadas en tarjetas, las cuales muestran el garzón, número de mesa, total consumido, fecha de ingreso, fecha de salida, observaciones y los productos pedidos. Estas tarjetas pueden ser ordenadas según el ingreso o salida, además de poder filtrar los pedidos a través de la información relevante. Además, cada tarjeta presenta dos botones: el botón de edición, que permite volver a abrir una mesa ya cerrada y cargar los productos pedidos, y el botón de eliminar, el cual permite borrar una transacción.

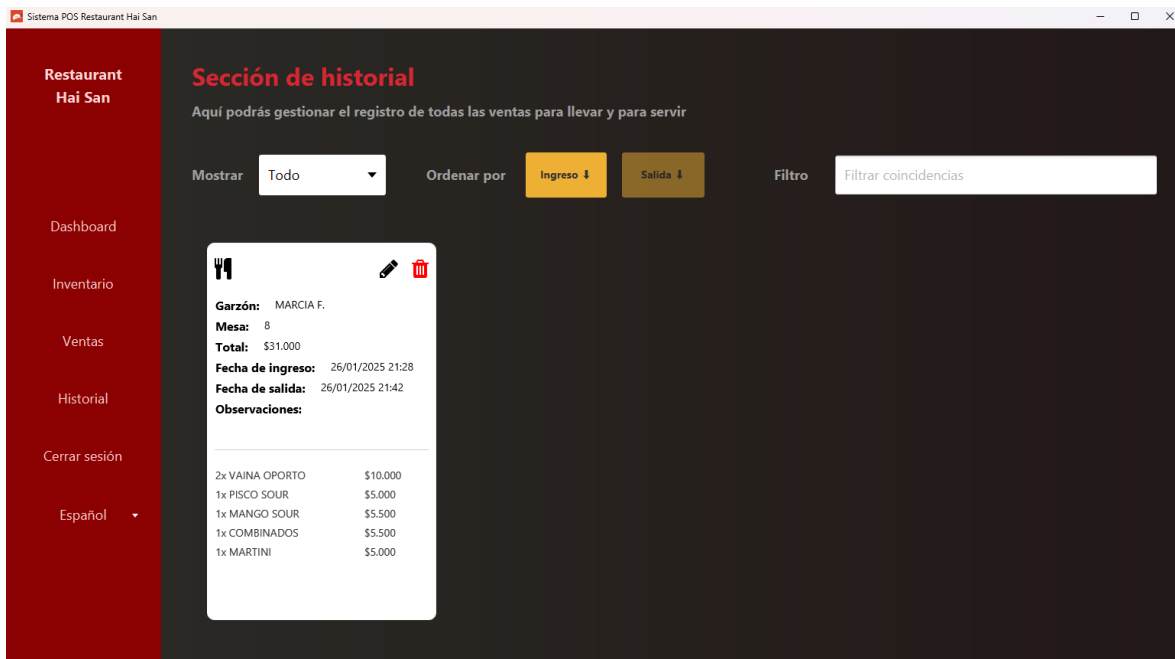


Figura 21: Vista del software, sección de historial.  
Fuente: Elaboración propia.

## 4.2. Periodo de prueba

Como parte de la validación de la solución, además de asegurarse del funcionamiento unitario de cada método utilizado (véase el apartado 3.8.4), se dedicó una semana para el monitoreo del software dentro de un entorno productivo.

Para analizar el rendimiento del sistema, se utilizó dentro del restaurante un ordenador Dell ThinkCentre M720q Tiny, un monitor Samsung y una impresora térmica genérica, cuyas especificaciones son detalladas en la Tabla 7.

| Componente       | Característica         |
|------------------|------------------------|
| Procesador       | Intel i5 8400T         |
| Tarjeta de video | Intel UHD Graphics 630 |
| Almacenamiento   | 512GB                  |
| Memoria RAM      | 8GB                    |
| Impresora        | Térmica Xprinter 80mm  |
| Monitor          | 1366x768, 60fps.       |

Tabla 7: Especificaciones técnicas del entorno tecnológico utilizado.  
Fuente: Elaboración propia.

Una vez ya instalado el entorno del sistema POS, se estableció un período de prueba de una semana, empezando el día 2 de septiembre de 2024 y finalizando el 8 del mismo mes.

|           | Cantidad de Pedidos diarios |       | Cantidad de Pedidos semanales |       | Retención promedio de mesas [min] | Primera mesa |             | Última mesa  |             | Mayor concentración de mesas |          |
|-----------|-----------------------------|-------|-------------------------------|-------|-----------------------------------|--------------|-------------|--------------|-------------|------------------------------|----------|
|           | Para llevar                 | Mesas | Para llevar                   | Mesas |                                   | Llegada [pm] | Salida [pm] | Llegada [pm] | Salida [pm] | Hora [pm]                    | Cantidad |
| Lunes     | 30                          | 28    | 30                            | 28    | 30                                | 13:37        | 13:54       | 19:57        | 21:17       | 14:21                        | 6        |
| Martes    | 31                          | 14    | 61                            | 42    | 31                                | 13:13        | 14:33       | 19:41        | 20:03       | 14:54                        | 4        |
| Miércoles | 37                          | 17    | 98                            | 59    | 36                                | 13:08        | 13:38       | 16:24        | 17:00       | 14:27                        | 9        |
| Jueves    | 22                          | 12    | 120                           | 71    | 38                                | 12:29        | 12:52       | 19:40        | 21:07       | 13:40                        | 3        |
| Viernes   | 41                          | 24    | 161                           | 95    | 37                                | 12:51        | 13:14       | 21:50        | 22:11       | 14:05                        | 8        |
| Sábado    | 42                          | 35    | 203                           | 130   | 35                                | 13:24        | 13:47       | 20:25        | 21:18       | 14:57                        | 12       |
| Domingo   | 49                          | 26    | 252                           | 156   | 34                                | 13:27        | 14:02       | 17:47        | 18:27       | 16:15                        | 9        |

Tabla 8: Monitoreo del funcionamiento del software en una etapa temprana, periodo 02 de septiembre - 08 de septiembre.

Fuente: Elaboración propia.

A partir de la **Tabla 8** se obtiene que en total se realizaron 408 pedidos, de los cuales 252 fueron del tipo para llevar y 156 fueron mesas. Cabe recalcar que dentro de estos datos no se añadieron las órdenes realizadas a través de PedidosYa, esto debido a que estos datos están almacenados dentro de su respectivo dispositivo.

Por otra parte, se desprende que se tuvo una retención promedio semanal de 34 minutos, es decir, cada mesa fue ocupada en promedio durante 34 minutos. Esta métrica se podría utilizar para evaluar qué tan rápido se realizó el servicio a cada cliente y, en caso de que múltiples días tengan un valor de retención promedio elevado, entonces se podría plantear el aumento de personal.

En cuanto a los pedidos, se observa una mayor concentración durante los fines de semana, siendo el sábado el día con más mesas activas simultáneamente, con un total de 12 mesas ocupadas a las 14:57 pm. Esta información podría ser relevante para ajustar los horarios de entrada y salida del personal, de modo que coincidan con los períodos de mayor demanda y se optimice el uso de los recursos humanos. Además, si en el futuro la cantidad de mesas ocupadas de forma simultánea tiende a alcanzar de manera recurrente la capacidad total del restaurante, esta métrica podría ser útil para justificar una eventual ampliación del local.

Por último, la primera mesa registrada fue a las 12:29 pm el día jueves y la última mesa registrada fue un viernes a las 22:11 pm. Esta información resulta útil, ya que el horario presentado en Google indica los siguientes horarios:

|                                           | Hora de inicio [pm] | Hora de cierre [pm] |
|-------------------------------------------|---------------------|---------------------|
| Lunes, Martes, Miércoles, Jueves, Viernes | 12:00               | 21:30               |
| Sábado                                    | 12:00               | 22:00               |
| Domingo                                   | 12:00               | 19:30               |

Tabla 9: Horarios de funcionamiento del local estudiado.

Fuente: [18].

Como se puede apreciar, la hora de la última mesa mostrada en la Tabla 8 del día viernes no es acorde a la hora mostrada en la Tabla 9. Debido a esto, se podría seguir monitoreando el comportamiento de la llegada de mesas y, en caso de seguir un patrón similar, modificar el horario de los días que no sean consistentes.

### 4.3. Prueba de estrés

En este apartado se analizó cómo se comporta el sistema desarrollado a medida que se incrementan parámetros relevantes para su uso. En este caso, se puso a prueba la vista de ventas, en donde se realizaron múltiples pruebas con una cantidad de mesas distintas.

Para este experimento, se ingresará una cantidad específica de mesas, las cuales serán inicializadas con productos. Para obtener la cantidad de productos asociadas a cada mesa, se calculó la cantidad promedio de productos pedidos por mesa en el tiempo de una semana, obteniendo una media de 3.62 productos por mesa, por lo que se iniciará cada mesa con 4 productos.

Por otra parte, para la generación de mesas se desarrolló un script que agregara una cantidad  $n$  de mesas, en donde  $n$  tomará los valores arbitrarios: 0, 10, 100 y 1000. De esta manera, se realizaron las siguientes iteraciones.

|             | Cantidad de mesas | Cantidad de productos |
|-------------|-------------------|-----------------------|
| Iteración 1 | 0                 | 4                     |
| Iteración 2 | 10                | 4                     |
| Iteración 3 | 100               | 4                     |
| Iteración 4 | 1000              | 4                     |

Tabla 10: Iteraciones realizadas durante el experimento.

Fuente: Elaboración propia.

De las cuales se obtuvieron los siguientes resultados:

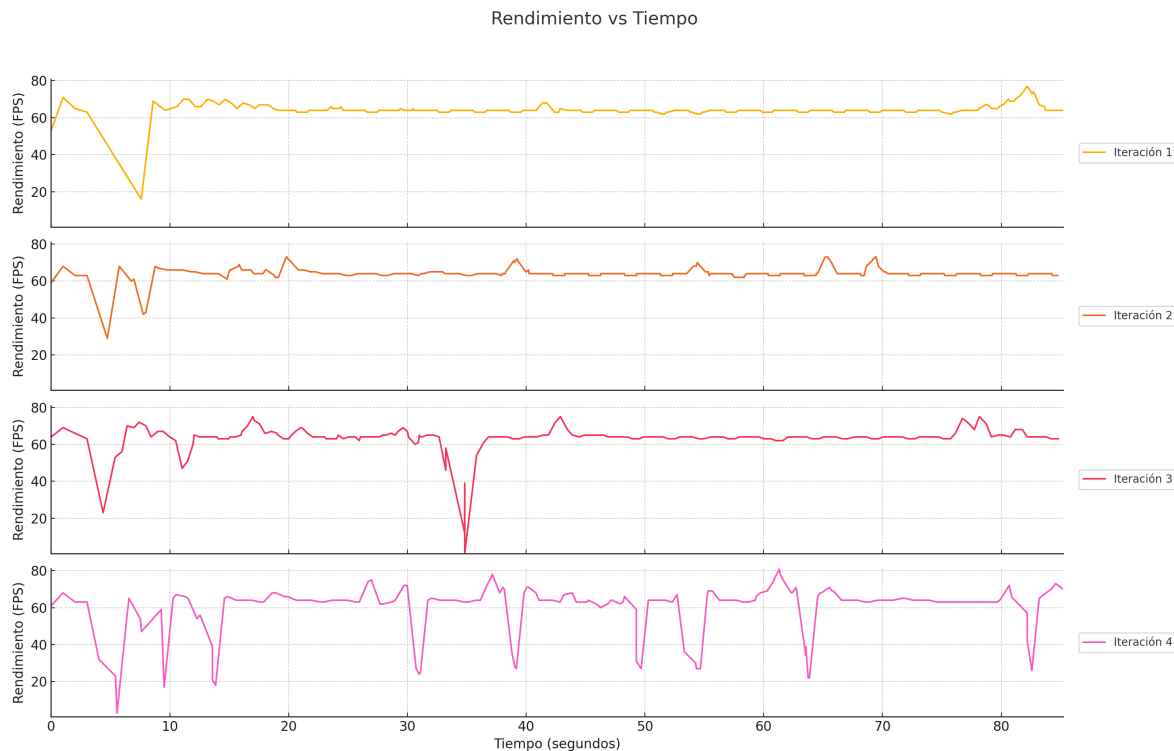


Figura 22: Gráfico comparativo rendimiento vs tiempo.  
Fuente: Elaboración propia.

A partir de la [Figura 22](#), se observa que el sistema presenta un funcionamiento estable al mantener 0 mesas activas, registrando un mínimo de 16 FPS y un único valor por debajo de los 60 FPS durante todo el experimento. Por otra parte, con 10 mesas activas, se obtuvo un mínimo de 29 FPS, además de dos caídas aisladas a 42 y 43 FPS, manteniéndose sobre los 60 FPS en el resto de las muestras.

Durante la iteración 3, se registró un mínimo de 1 FPS, con algunas caídas menores a 60 FPS que representaron un 4.59 % del total de muestras. En la iteración 4, el mínimo fue de 3 FPS, y se observaron múltiples caídas por debajo de los 60 FPS, correspondientes al 13.45 % de las muestras totales. Si bien este valor es relevante, no resulta preocupante en un escenario realista, dado que no es esperable que un restaurante tenga activas 1000 mesas simultáneamente.

En consecuencia, se puede concluir que, aunque el rendimiento del sistema disminuye a medida que aumenta la cantidad de mesas activas, este se mantiene en niveles aceptables entre las 10 y 100 mesas, lo que corresponde a una carga realista en este caso de prueba. En el peor de los casos, la reducción de rendimiento alcanzaría un 4.59 %, valor que no compromete la usabilidad del sistema en condiciones normales de operación.

#### 4.4. Impacto del sistema generado en el negocio

Antes de la implementación del sistema POS desarrollado, el cajero registraba los pedidos manualmente utilizando una hoja de papel con formato de lista. Esta lista incluía columnas para la cantidad, el nombre del producto (preimpreso) y su precio total. Cabe señalar que no era factible incluir toda la carta del restaurante en este formato, tanto por limitaciones de tamaño de hoja como por razones de legibilidad.

Por ello, el listado contenía únicamente una selección de productos, además de un apartado para escribir hasta tres productos adicionales no incluidos en la lista o productos fuera de menú, una sección específica para bebestibles y otra destinada al cálculo del subtotal, la propina sugerida y el total. En la **Tabla 11** se muestra una simulación del formato utilizado anteriormente por el restaurante para registrar ventas.

| RESTAURANT HAI SAN |            |        | Día  | Mes                          | Año    |
|--------------------|------------|--------|------|------------------------------|--------|
| Cant               | Detalle    | Precio | Cant | Detalle                      | Precio |
|                    | Producto 1 |        |      | Producto n+1                 |        |
|                    | Producto 2 |        |      | Producto n+2                 |        |
|                    | Producto 3 |        |      | Producto n+3                 |        |
|                    | Producto 4 |        |      | Producto n+4                 |        |
|                    | Producto n |        |      | Producto n+5                 |        |
|                    |            |        |      | Producto n+6                 |        |
|                    |            |        |      | Producto n+7                 |        |
|                    |            |        |      | Producto n+8                 |        |
|                    | Vinos      |        |      | Producto n+9                 |        |
|                    |            |        |      | Producto n+10                |        |
|                    | Bebidas    |        |      | <b>Subtotal</b>              |        |
|                    | Cervezas   |        |      | <b>10 % Propina sugerida</b> |        |
|                    |            |        |      | <b>Total</b>                 |        |

Tabla 11: Simulación hoja de detalle utilizada anteriormente dentro del negocio.

Fuente: Elaboración propia.

Al analizar la **Tabla 11** se evidencian diversas falencias, siendo la principal la dificultad que enfrentaban los clientes para leer algunos detalles. Inicialmente, el cliente debía revisar toda la hoja para encontrar los productos solicitados. Además, cuando el cliente agregaba o eliminaba productos ya pedidos, el cajero se veía obligado a sobrescribir los valores de cantidad y precio, lo que hacía el detalle cada vez menos legible.

Cabe destacar que este formato carecía de un apartado específico para agregar excepciones, como descuentos u observaciones, por lo que era necesario escribir sobre apartados no ocupados dentro del detalle para agregarlos. Finalmente, en los casos donde el cajero come-

tía errores en la suma de productos, debía modificar manualmente los valores de subtotal, propina sugerida y total, lo que empeoraba la legibilidad de la cuenta.

Con la implementación del sistema POS, estos problemas se resuelven efectivamente, ya que permite almacenar múltiples pedidos donde se pueden modificar los productos registrados antes de finalizar la cuenta. Asimismo, facilita la incorporación de excepciones como descuentos, observaciones o productos fuera de menú, los cuales se reflejan automáticamente en un detalle estandarizado. Este sistema elimina el error humano durante el proceso y mejora significativamente la experiencia tanto de los clientes como de los usuarios del sistema.

Por otra parte, esta metodología también presentaba falencias en términos temporales, ya que por cada producto se debía anotar su cantidad y su precio. En caso de que no se supiera el precio del producto, se debía consultar la carta, lo que era bastante ineficiente en temas de tiempo. Por otra parte, había ocasiones en donde, al realizar múltiples sobrescrituras de algún producto, se debía realizar el detalle nuevamente por falta de espacio, lo que suponía una carga adicional para el cajero. Por último, y al igual que en la sección anterior, la necesidad de tener que sumar todos los productos de manera manual al momento de realizar la cuenta también suponía una ineficacia en el uso del tiempo.

Para verificar si el sistema POS optimizó el uso de tiempo a la hora de registrar una venta, se realizó un breve experimento con el usuario, en donde se le solicitó al cajero registrar una venta como se hacía anteriormente, utilizando los mismos detalles utilizados con anterioridad y realizando 2 pedidos ficticios por cantidad de producto único. De esta manera, la toma de tiempo contemplaba:

- Ingreso de la fecha.
- Ingreso del número de mesa.
- Ingreso de productos, tanto su cantidad como su precio.
- Búsqueda del precio de algún producto utilizando la carta (en caso de ser necesario).
- Cálculo de la cuenta.

Y de lo cual se obtuvieron los siguientes resultados:

| Productos | Tiempo pedido 1 [s] | Tiempo pedido 2 [s] | Promedio [s] |
|-----------|---------------------|---------------------|--------------|
| 1         | 10.32               | 10.92               | 10.62        |
| 2         | 20.44               | 20.99               | 20.71        |
| 3         | 26.83               | 28.69               | 27.76        |
| 4         | 30.65               | 49.75               | 40.20        |
| 5         | 47.89               | 57.83               | 52.86        |
| 6         | 69.33               | 81.79               | 75.56        |
| 7         | 72.61               | 96.17               | 84.39        |

Tabla 12: Tiempos de registro manual de ventas según la cantidad de productos.  
Fuente: Elaboración propia.

Por otra parte, para calcular el tiempo empleado en registrar una venta de un pedido utilizando el sistema POS se desarrolló un temporizador dentro de la sección de venta de pedidos para mesa, en donde se consideró el tiempo desde que se abre una mesa hasta que se realiza la cuenta. Es importante destacar que dicho temporizador aumentaba su tiempo solamente si el usuario se encontraba dentro de la vista del pedido; en caso contrario, se pondría en pausa. De esta manera, se destinó desde el 9 hasta el 15 de septiembre para la obtención de muestras, en donde se consiguieron los siguientes resultados:

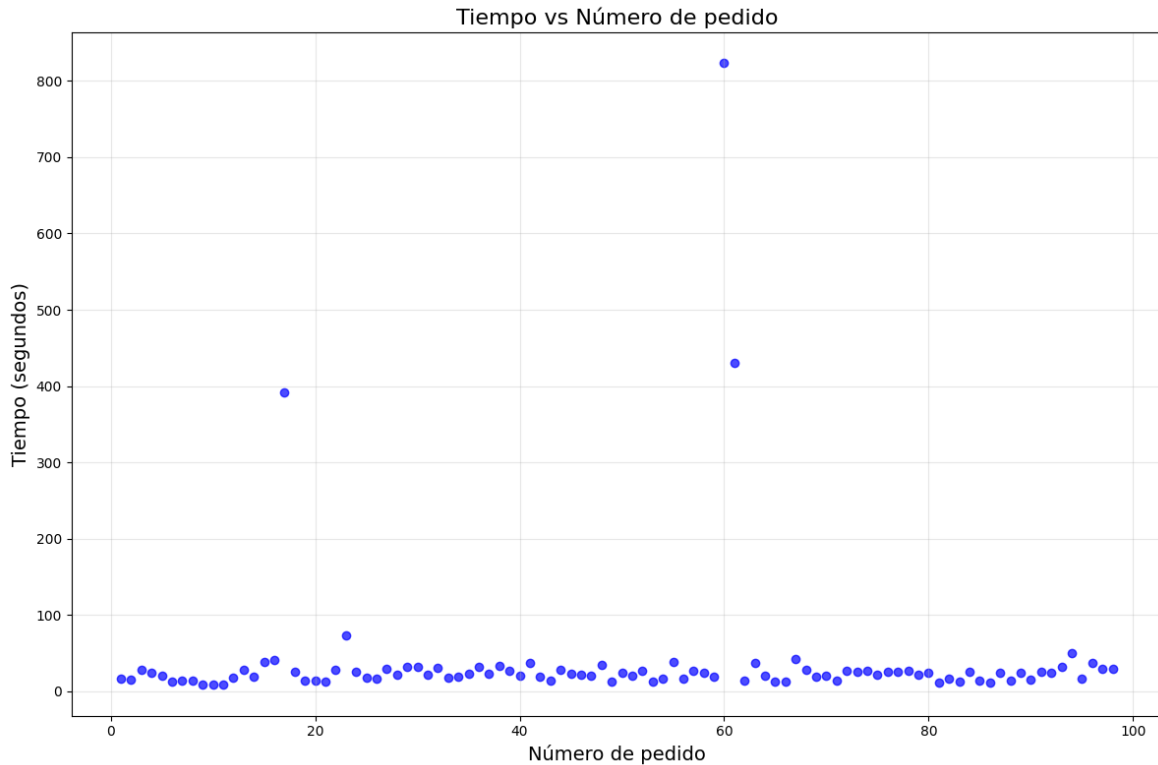


Figura 23: Tiempo de registro por pedido utilizando el sistema POS.  
Fuente: Elaboración propia.

Tal como se aprecia en la [Figura 23](#), los tiempos registrados presentan una alta variabilidad, con valores que fluctúan entre 0 y 100 segundos, y algunos casos extremos que alcanzan entre 300 y 850 segundos. Esta dispersión se explica, en parte, porque en ciertos momentos el usuario dejaba abierta la pestaña de una mesa sin interactuar activamente con el sistema, lo que provocaba una acumulación de tiempo no representativa del uso real. Considerando esta situación, se optó por excluir estos datos extremadamente atípicos y utilizar diagramas de caja, con los cuales se podrá visualizar de manera más representativa la mediana, promedio, valores atípicos (en caso de haber) y cómo se distribuyen los datos en cada rango de cantidad de productos.

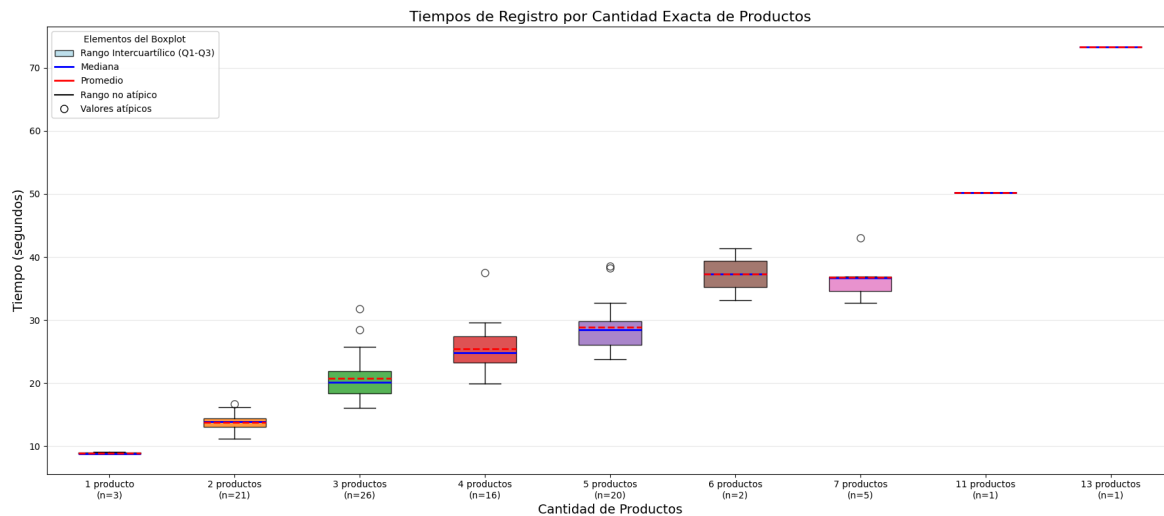


Figura 24: Diagramas de caja tiempo de registro por cantidad de productos.  
Fuente: Elaboración propia.

Anexado a estos diagramas de caja, se tienen las siguientes métricas:

| Productos | Pedidos | Promedio [s] | Mínimo [s] | Máximo [s] | Desviación estándar [s] |
|-----------|---------|--------------|------------|------------|-------------------------|
| 1         | 3       | 8.9          | 8.8        | 9.1        | 0.14                    |
| 2         | 21      | 13.79        | 11.2       | 16.7       | 1.3                     |
| 3         | 26      | 20.72        | 16         | 31.8       | 3.72                    |
| 4         | 16      | 25.4         | 19.9       | 37.5       | 4                       |
| 5         | 20      | 28.88        | 23.8       | 38.5       | 3.89                    |
| 6         | 2       | 37.25        | 33.1       | 41.4       | 4.15                    |
| 7         | 5       | 36.78        | 32.7       | 43         | 3.47                    |
| 11        | 1       | 50.2         | 50.2       | 50.2       | 0                       |
| 13        | 1       | 73.3         | 73.3       | 73.3       | 0                       |

Tabla 13: Tiempos de registro mediante el uso del sistema POS según cantidad de productos.  
Fuente: Elaboración propia.

De esta manera, al comparar la [Tabla 12](#) con la [Tabla 13](#) se puede evidenciar que:

En el caso de los pedidos que contenían un solo producto único, se analizaron 3 registros, los cuales presentaron un tiempo promedio de 8.90 segundos y una desviación estándar de 0.14 segundos. Estos resultados reflejan una ejecución consistente en el uso del sistema POS. Al compararlo con la metodología manual anterior, se evidenció una mejora de 1.72 segundos en promedio a favor del sistema desarrollado, lo que corresponde a un porcentaje de mejora de 16.2 %.

Para los pedidos con dos productos únicos, se evaluaron 21 registros. El tiempo promedio fue de 13.79 segundos, con una desviación estándar de 1.30 segundos. Si bien se observó un aumento en la desviación estándar respecto al grupo anterior, este valor aún se considera dentro de un margen razonable. Al comparar este desempeño con la metodología anterior, se evidenció una mejora de 6.92 segundos, lo que representa una reducción del 33.41 % en el tiempo de registro.

Para los pedidos con tres productos, se analizó una muestra de 26 registros, alcanzando un promedio de 20.72 segundos y una desviación estándar de 3.72 segundos. En comparación con los grupos anteriores, se observa un aumento significativo en la variabilidad, explicado en parte por la presencia de dos valores atípicos que superan los límites del diagrama de caja. Estos valores podrían atribuirse a distracciones del usuario o a la inclusión de productos fuera de menú, los cuales tienden a aparecer con mayor frecuencia en pedidos de tres productos o más. Este tipo de productos requiere una mayor interacción por parte del usuario, ya que deben ser ingresados manualmente mediante el teclado, una tarea con la que el usuario aún no está completamente familiarizado. Esta diferencia en la complejidad genera una alta variabilidad entre los pedidos que contienen productos fuera de menú y aquellos que no, lo que impacta directamente en la desviación estándar. No obstante, a pesar de esta variabilidad, el uso del sistema supuso una mejora de 7.04 segundos en promedio frente al procedimiento manual, lo que evidencia una mejora de 25.36 %.

En cuanto a los pedidos con cuatro productos únicos, se consideraron 16 registros. Estos presentaron un promedio de 25.40 segundos y una desviación estándar de 4.00 segundos, incluyendo un valor atípico por sobre el máximo. Dicho valor podría explicarse por las mismas condiciones observadas en los pedidos con tres productos. A partir de este grupo y en los que siguen, todas las desviaciones estándar superan los 3 segundos, lo que podría reforzar la hipótesis de que los productos fuera de menú influyen en la dispersión de los tiempos de registro. En este grupo, la comparación con la metodología tradicional evidenció una mejora de 14.80 segundos en promedio o un 36.81 %.

Para los pedidos compuestos por cinco productos únicos, se analizaron 20 registros. El tiempo promedio fue de 28.88 segundos, con una desviación estándar de 3.89 segundos y la presencia de dos valores atípicos por sobre el máximo. Al comparar estos resultados con el registro manual, se obtuvo una mejora significativa de 23.98 segundos, lo que redujo en un 45.36 % el tiempo de registro de venta.

Respecto a los pedidos con seis productos únicos, se evaluaron 2 registros, alcanzando un promedio de 37.25 segundos y una desviación estándar de 4.15 segundos. En comparación con la metodología anterior, el tiempo de registro se redujo en 38.31 segundos o un 50.71 %.

En el caso de los pedidos que incluyeron siete productos únicos, se registraron 5 casos, con un promedio de 36.78 segundos y un valor atípico identificado. La mejora respecto al método manual fue de 47.61 segundos, lo que supuso una reducción del 56.41 %.

Para los pedidos que incluyeron entre 8 y 13 productos únicos no se realizó un análisis com-

parativo, debido a la ausencia de datos suficientes, ya sea en el uso del sistema POS o en el registro manual de venta.

A continuación, se presenta una síntesis gráfica de la información analizada.

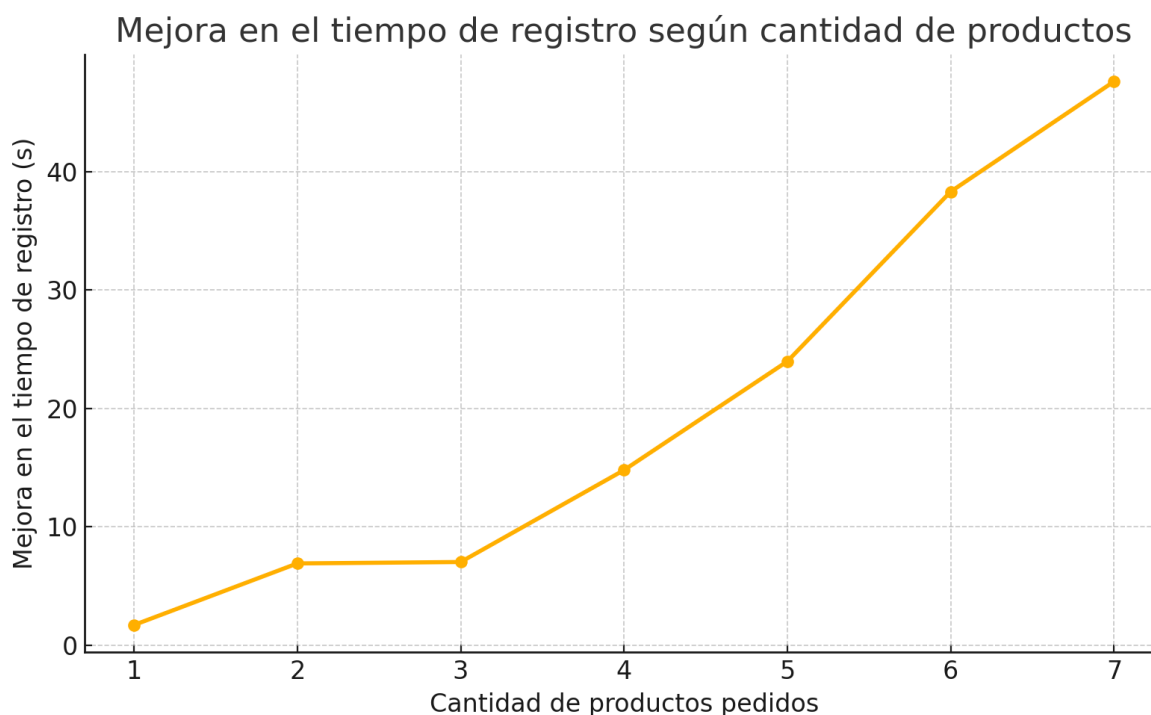


Figura 25: Mejora en tiempo de registro según cantidad de productos únicos.  
Fuente: Elaboración propia.

Finalmente, al observar el conjunto total de pedidos y la [Figura 25](#), se evidencia una tendencia clara: a mayor cantidad de productos únicos, mayor es la optimización de tiempo obtenida mediante el uso del sistema. Cabe destacar que el usuario aún no demuestra un dominio completo del teclado ni del sistema, pues estas pruebas fueron realizadas en etapas tempranas tras la implementación del software, por lo que es razonable suponer que, conforme el usuario vaya utilizando el sistema POS y acostumbrándose al uso del teclado, la frecuencia de valores atípicos disminuirá y la variabilidad en los tiempos tenderá a reducirse, mejorando aún más los tiempos de uso del sistema POS.

Por otra parte, la metodología antigua presentaba problemas significativos con la trazabilidad de ventas, debido a que los detalles físicos corrían el riesgo de extraviarse, dañarse o desecharse accidentalmente, lo que suponía un obstáculo importante al momento de requerir consultas históricas. Asimismo, la ausencia de una cronología específica en el registro de las mesas constituía otra dificultad para verificar con exactitud el orden de llegada de los clientes, ya que, si bien los detalles incluían día, mes y año, carecían de información sobre la hora exacta de llegada y salida. El sistema POS ha solucionado eficazmente estas limitaciones

al proporcionar una persistencia de datos consistente, donde cada orden se almacena automáticamente con su registro cronológico, además de habilitar funcionalidades de filtrado y búsqueda de órdenes según diversos parámetros, mejorando así la trazabilidad de órdenes.

Finalmente, la metodología utilizada anteriormente por el negocio carecía de la capacidad para proporcionar métricas precisas y relevantes para la gestión del negocio. Por ejemplo, los administradores de negocio podían tener una noción aproximada sobre cuáles fueron los productos más vendidos o en qué franjas horarias se concentraba la mayor cantidad de pedidos, pero sin la exactitud necesaria para tomar decisiones fundamentadas. En este sentido, el despliegue de métricas e indicadores de ganancia que ofrece el sistema POS ha permitido al negocio validar con precisión las problemáticas anteriormente mencionadas, además de facilitar la comparación del desempeño actual con períodos anteriores, proporcionando así una base sólida para la toma de decisiones.

#### 4.5. Impacto de las prácticas ágiles

Como se mencionó en secciones anteriores, para el desarrollo de este sistema se implementaron distintas prácticas asociadas a metodologías ágiles, las cuales fueron previamente adaptadas a un contexto donde el proyecto fue realizado por una sola persona y en colaboración con una microempresa sin experiencia técnica. A pesar de esas limitaciones, estas prácticas resultaron ser clave para avanzar de forma ordenada, mantener la continuidad del trabajo y permitir una validación progresiva de funcionalidades.

Uno de los principales impactos positivos derivados de la implementación de estas prácticas ágiles fue la mejora continua del software a través de iteraciones, donde el usuario podía comprobar periódicamente el estado del sistema y, en caso de requerir algún cambio o adición, comunicarlo en una etapa temprana del desarrollo. Tal y como se observa en los apartados [3.8.2](#), [3.8.3](#) y [3.8.4](#), en donde el cliente fue capaz de agregar y modificar funcionalidades de acuerdo a necesidades que inicialmente no habían sido identificadas con claridad, permitiendo así una adaptación progresiva del sistema a los requerimientos reales del negocio.

Por otro lado, la estandarización del código y su adecuada documentación contribuyeron significativamente a la construcción de un software más robusto y escalable, lo que permitió implementar de manera más eficiente las funcionalidades requeridas. Además, las pruebas unitarias fueron fundamentales para detectar errores previos al paso a producción. Por ejemplo, ayudaron a identificar un fallo en la generación del identificador de órdenes, el cual no se reiniciaba correctamente al cambiar de semana. De manera similar, permitieron detectar de forma anticipada un comportamiento incorrecto al ordenar y filtrar las órdenes en el historial, donde el sistema no respondía como se esperaba al aplicar varias operaciones de filtrado y ordenación de forma consecutiva.

Por su parte, la implementación del tablero Kanban permitió representar las historias de

usuario descritas en la [Subsección 3.7](#). Como se mencionó anteriormente, se utilizó Trello como herramienta para visualizar dicho tablero. De manera predeterminada, este no cuenta con un sistema de priorización, por lo que se integró la extensión *Card Priority by Screenful* [19], la cual permitió clasificar las tareas según su nivel de urgencia.

Esta integración facilitó la organización y selección visual de las tareas a ejecutar en cada iteración. Tal como se muestra en las Figuras 11, 12, 13 y 14, en un inicio se priorizaron tareas con etiquetas “*Highest*” y “*High*”, correspondientes a aquellas que aportaban mayor valor al negocio. A medida que avanzaban las iteraciones y se completaban dichas tareas, se priorizaron tareas con una clasificación “*Medium*” y, finalmente, se realizaron las tareas con prioridad “*Low*”.

Por otra parte, la implementación de un límite de trabajo en progreso de dos tareas simultáneas permitió observar distintos esquemas de trabajo. En los casos en que se abordó una sola tarea a la vez, se evidenció una mayor concentración al no tener que alternar entre contextos distintos, y se redujeron los bloqueos al enfocarse en una única actividad. En cambio, cuando se trabajó en dos tareas de forma simultánea pero con elementos en común, como el uso compartido de métodos, propiedades o clases, se observó una reducción en el tiempo de desarrollo. Esta modalidad permitió reutilizar conocimientos y recursos de forma inmediata entre ambas tareas, evitando la duplicación de esfuerzos. Al considerar en conjunto los requisitos y parámetros de dichas tareas, fue posible diseñar componentes, métodos y clases reutilizables, lo que reforzó el principio “*Don’t Repeat Yourself*” (DRY) [20] propio del desarrollo de software y promovió un desarrollo más eficiente y cohesivo.

Para llevar esta práctica a la herramienta utilizada, fue necesario implementar soporte adicional. Al igual que en el caso de la priorización de tareas, Trello no cuenta con una funcionalidad nativa para limitar la cantidad máxima de tareas por columna. Por ello, se integró el complemento “*Kanban WIP by Corello*” [21], con el objetivo de restringir la columna “En progreso” al límite establecido. Cabe destacar que, si bien se estableció un límite máximo de dos tareas en progreso, se prefirió trabajar en una sola tarea a la vez. Solo en casos específicos, donde las tareas presentaban una relación directa y compartían lógica de código o de negocio, se prefirió el desarrollo simultáneo, con el fin de aprovechar sinergias y minimizar el impacto de los cambios de contexto. Por ejemplo, se realizaron de manera simultánea las vistas de venta específicas para mesas y para llevar, puesto que comparten el mismo archivo de código, en donde algunos campos, botones y métodos eran llamados únicamente si el pedido era de un tipo u otro. Otro ejemplo en el que se optó por la multitarea fue durante el desarrollo de las funcionalidades de edición y eliminación de productos dentro del inventario, ya que ambas se integraron en una misma tabla y comparten una lógica similar: ejecutar una operación de tipo CRUD luego de un evento accionador, como el darle clic a un ícono o darle doble clic a una celda.

El Daily Standup, por su parte, sirvió para planificar diariamente las funcionalidades a desarrollar, al contextualizar de manera breve y concisa lo desarrollado en la jornada de desarrollo anterior. Asimismo, sirvió para identificar bloqueos de manera temprana, al tener que registrar de manera diaria dentro de la bitácora las dificultades presentadas durante la

implementación del sistema.

Por último, se consideró que el uso del ritmo sostenible no fue del todo relevante, ya que, si bien sirvió para declarar un máximo semanal, en ningún momento del desarrollo se superó dicho límite ni se cuestionó su aumento, por lo que no afectó de manera activa en el desarrollo del software.

## CAPÍTULO 5

### CONCLUSIONES

La presente memoria abordó el desafío de desarrollar e implementar un sistema de Punto de Venta (POS) en un entorno microempresarial, específicamente en el restaurante Hai San, enfrentando restricciones económicas, limitaciones de tiempo y una alta incertidumbre en cuanto a los requisitos funcionales. En este contexto, se propuso como eje central la adopción de prácticas ágiles seleccionadas cuidadosamente, adaptadas a un marco de trabajo de un solo desarrollador y a las particularidades propias del negocio gastronómico.

En un principio, se analizó y escogió el enfoque de desarrollo, en donde se estableció el desarrollo de una aplicación de escritorio en base a sus ventajas comparativas, tomando en consideración las restricciones de infraestructura y recursos, y la experiencia previa del desarrollador.

Luego, durante el desarrollo del sistema, se demostró que es factible aplicar prácticas ágiles incluso en escenarios de baja escala, siempre que se ajusten las prácticas a las condiciones reales del proyecto.

Si bien la mayoría de las prácticas aplicadas resultaron beneficiosas, una de ellas tuvo un impacto neutro y no aportó ventajas perceptibles en el contexto del proyecto. En particular, la limitación del trabajo en progreso, el uso de iteraciones, la implementación de un Tablero Kanban y la planificación diaria mediante Daily standups contribuyeron significativamente a diversos aspectos del proceso de desarrollo: se promovió la mejora continua, se facilitó la organización y planificación de tareas, se redujeron los tiempos de desarrollo y se logró detectar errores y bloqueos de forma anticipada. Estas prácticas también favorecieron la construcción de un software más robusto y con proyección de escalabilidad. En contraste, la práctica de mantener un ritmo sostenible no evidenció un impacto tangible, posiblemente debido a las características particulares del proyecto y la escala reducida del equipo de desarrollo.

Por su parte, para evaluar el impacto del sistema desarrollado en el funcionamiento del negocio, se realizaron distintos experimentos orientados al análisis de información y toma de decisiones en base a esta. En una primera instancia, se destinó una semana a la recolección de información en un entorno productivo. Durante ese período se registraron un total de 408 pedidos, en donde se analizaron variables relevantes como la retención promedio de mesas, la hora de llegada y salida de la primera y última mesa del día y la cantidad de pedidos diarios, con el objetivo de identificar patrones de comportamiento y contrastarlos con información existente del negocio. A través de esta fase de prueba, se detectaron anomalías en la entrada y salida de mesas en horarios que figuraban como fuera de servicio en Google. Asimismo, se evidenció una mayor concentración de pedidos durante los fines de semana, donde tanto las ventas para llevar como las mesas atendidas superaron consistentemente las cifras de los días de semana, cuyo aumento no reflejó de manera negativa en la reten-

ción promedio. Otro hallazgo relevante fue que los días domingo, la mayor concentración de mesas se producía en horarios más tardíos en comparación con el resto de la semana.

Si bien estos datos obtenidos ofrecieron indicios preliminares de tendencias relevantes, se recomendó extender el período de observación con el fin de confirmar patrones más consistentes que puedan respaldar decisiones estratégicas, como ajustes en los turnos de trabajo o refuerzos de personal en horarios específicos.

En una segunda instancia, se midió el tiempo de registro de ventas, comparando el método tradicional manual con el uso del nuevo sistema POS. Para ello, se simularon 14 ventas utilizando el método manual y se analizaron 95 registros reales generados a través del sistema POS, ambos agrupados por la cantidad de productos únicos involucrados en cada transacción. Los resultados demostraron una reducción significativa en los tiempos de registro al utilizar el sistema desarrollado, evidenciando mejoras consistentes en eficiencia operativa. En los casos más favorables, se observó una mejora de hasta un 56.41 % en el rendimiento, en donde se debe considerar que este experimento fue en épocas tempranas de la implementación del sistema, por lo que se estima que el tiempo de registro de una venta disminuya aún más.

Además, con el objetivo de validar el correcto funcionamiento del sistema bajo condiciones de alta carga, se estimó la cantidad promedio de productos pedidos por mesa, obteniendo una media de 4 productos por mesa. A partir de esta estimación, se realizaron simulaciones con escenarios de 0, 10, 100 y 1000 mesas activas, cada una con 4 productos, con el fin de evaluar el impacto en el rendimiento del software.

Los resultados de esta prueba de estrés indicaron que, dentro de un rango de 0 a 10 mesas activas, el sistema se comporta de forma estable y fluida, sin presentar caídas significativas de rendimiento luego de los primeros 10 segundos. En el caso de 100 mesas activas, se observaron caídas puntuales de hasta 1 FPS, aunque estas fueron instancias aisladas, ya que solo el 4,59 % de las muestras registraron valores por debajo de los 60 FPS.

Por otro lado, el escenario de 1000 mesas concurrentes presentó una disminución del 13.45 % en el rendimiento, lo cual constituye una degradación considerable. Sin embargo, este escenario no se ajusta a la realidad operativa del negocio, por lo que no representa una preocupación en la práctica.

Considerando que el restaurante opera habitualmente entre 10 y 100 mesas, se concluye que el desempeño del sistema es aceptable. Incluso en el caso más exigente dentro de ese rango, la pérdida máxima de rendimiento fue mínima y no compromete la experiencia de uso. Además, es altamente improbable que en el futuro cercano se alcance un volumen de operación que supere las 100 mesas activas de manera simultánea.

En cuanto a los límites y proyecciones de este proyecto, se plantean diversas ramas de mejora, las cuales dependen de las necesidades evolutivas y cambios dentro de la infraestructura o equipo de trabajo.

Como primera línea de mejora, se propone el desarrollo de una aplicación móvil que extienda las funcionalidades del sistema existente. Esta permitiría a los garzones registrar directamente las ventas desde sus dispositivos móviles, los cuales estarían conectados al sistema central. De este modo, se eliminaría la necesidad de que el cajero registre manualmente las ventas en el sistema, agilizando el proceso de atención y reduciendo la carga operativa del cajero o administrador de negocio, el cual podría enfocarse en otras tareas, como la atención y recepción de clientes.

Para ello, sería necesario reubicar el módem actualmente instalado y, de ser requerido, incorporar repetidores de señal que permitan asegurar una conectividad estable en todo el local. Además, se debería capacitar tanto a garzones como a cajeros en el uso de la aplicación móvil, ya que, aunque están familiarizados con el uso cotidiano de estos dispositivos, muchos de los garzones presentan dificultades para operarlos de forma ágil y precisa, debido principalmente a su edad avanzada.

Como segunda línea de mejora, se propone una integración con la carta. Actualmente, el repositorio que contiene el sitio web de la carta del negocio se encuentra alojado en *GitHub*, el cual, cada vez que se hace un cambio dentro de su archivo *HTML* asociado, se hace el despliegue de la carta de manera automática dentro del dominio específico. Por otra parte, la carta se modifica y exporta en formato PDF a través de la plataforma *Canvas*.

De esta manera, se propone la capacidad de cambiar la carta y reflejar los cambios dentro del dominio en donde se aloja esta misma cada vez que se necesite realizar un cambio dentro del inventario.

Finalmente, con el objetivo de centralizar aún más las funcionalidades del restaurante dentro del sistema, se plantea la posibilidad de integrar la plataforma PedidosYa mediante su API. Esta integración permitiría visualizar las órdenes generadas en dicha plataforma directamente desde el sistema POS, así como aceptar, rechazar o actualizar el estado de los pedidos en tiempo real. Además, facilitaría la incorporación automática de estas ventas a los indicadores de ganancia y rendimiento del negocio.

Esta funcionalidad no solo mejoraría la trazabilidad y eficiencia operativa, sino que también mitigaría los problemas de rendimiento observados en el equipo proporcionado por PedidosYa, el cual presenta ralentizaciones en momentos de alta demanda. Al consolidar toda la lógica operativa en un único sistema, se reduciría la carga manual para los administradores, evitando la necesidad de realizar cálculos externos y permitiendo una gestión más eficiente y centralizada.

## REFERENCIAS BIBLIOGRÁFICAS

- [1] Balasubramaniam Ramesh, Lan Cao, and Richard Baskerville. Agile requirements engineering practices and challenges: an empirical study. *Information Systems Journal*, 20(5):449–480, 2010.
- [2] Rebecca Sandstø and Cornelia Reme-Ness. Agile practices and impacts on project success. 2021.
- [3] Lubna Siddique and Bassam A Hussein. Enablers and barriers to customer involvement in agile software projects in norwegian software industry: The supplier's perspective. *The Journal of Modern Project Management*, 7(2), 2019.
- [4] Wikipedia contributors. Software crisis — wikipedia, the free encyclopedia, 2025.
- [5] Hironori Washizaki, editor. *Guide to the Software Engineering Body of Knowledge (SWE-BOK), Version 4.0*. IEEE Computer Society, 2024.
- [6] Kent Beck, M Beedle, A Van Bennekum, A Cockburn, W Cunningham, M Fowler, J Grenning, J Highsmith, A Hunt, R Jeffries, et al. The agile manifesto. agile alliance. *References-Scientific Research Publishing*, 2001.
- [7] Digital.ai. 15th state of agile report. <https://itnove.com/wp-content/uploads/2021/07/15th-state-of-agile-report.pdf>, July 2021.
- [8] ClickUp. T-shirt sizing: A guide to sizing agile story points. <https://clickup.com/blog/t-shirt-sizing/>, 2023.
- [9] Asana. ¿qué es el planning poker y cómo se usa? <https://asana.com/es/resources/planning-poker>, 2023.
- [10] Scrum Manager. Bucket system. [https://www.scrummanager.com/bok/index.php/Bucket\\_System](https://www.scrummanager.com/bok/index.php/Bucket_System), 2023.
- [11] Daily.dev. Dot voting: Técnica de priorización en agile. <https://daily.dev/es/blog/dot-voting-in-agile-prioritization-technique>, 2023.
- [12] Trello. Trello: Organize anything, together. <https://trello.com/es>, 2024.
- [13] Journey.Cloud. Journey: Your private and secure digital diary, 2024.
- [14] Sun Microsystems, Inc. *Java Code Conventions*, 1999.
- [15] JetBrains. Java best practices. <https://blog.jetbrains.com/idea/2024/02/java-best-practices/>, February 2024.

- [16] Scrum.org. Practical fibonacci: A beginner's guide to relative sizing. <https://www.scrum.org/resources/blog/practical-fibonacci-beginners-guide-relative-sizing>, 2021.
- [17] Asana. La matriz de eisenhower: cómo organizar tus tareas y prioridades, 2024.
- [18] Google. Restaurant hai san, 2025.
- [19] Screenful. Card priority by screenful. <https://trello.com/power-ups/5d40dbf16b5f44535df106d1/card-priority-by-screenful>, 2019.
- [20] Wikipedia contributors. Don't repeat yourself — Wikipedia, the free encyclopedia, 2025.
- [21] Corrello. Kanban wip by corrello — trello power-up, 2025.