UNIVERSIDAD TÉCNICA DE COTOPAXI

FACULTAD DE CIENCIAS DE LA INGENIERÍA Y APLICADAS

CARRERA DE SISTEMAS DE LA INFORMACIÓN

PROPUESTA TECNOLÓGICA

DESARROLLO DE UNA APLICACIÓN WEB PROGRESIVA PARA LA GESTIÓN DE PEDIDOS Y COMUNICACIÓN EN TIEMPO REAL DEL CAFÉ RESTAURANTE "PRODUCTOS CARLOS GERARDO" DEL TENA

Propuesta Tecnológica previo a la obtención del Título
de Ingeniera en Sistemas de la Información

AUTOR(ES):
Katherin Jamilep Pico Jimenez

TUTOR:
Mgs. Victor Hugo Medina Matute

LATACUNGA – ECUADOR

MARZO 2026

Latacunga, marzo 2026

DECLARACIÓN DE AUTORÍA

Yo, PICO JIMENEZ KATHERIN JAMILEP, con cedula de ciudadanía No. 1550252124 declaro ser autora de la **PROPUESTA TECNOLÓGICA: "DESARROLLO DE UNA APLICACIÓN WEB PROGRESIVA PARA LA GESTIÓN DE PEDIDOS Y COMUNICACIÓN EN TIEMPO REAL DEL CAFÉ RESTAURANTE "PRODUCTOS CARLOS GERARDO" DEL TENA"**, siendo el Mgs. Victor Hugo Medina Matute tutor del presente trabajo de titulación; y eximo expresamente a la Universidad Técnica de Cotopaxi y a sus representantes legales de posibles reclamos o acciones legales.

Además, certifico que las ideas, conceptos, procedimientos y resultados vertidos en el presente trabajo de titulación, son de mi exclusiva responsabilidad.

Katherin Jamilep Pico Jimenez
CC. 1550252124

Latacunga, marzo 2026

AVAL DEL TUTOR DEL PROYECTO DE PROPUESA TECNOLÓGICA

En calidad de Tutor de la Propuesta Tecnológica sobre el título: **"DESARROLLO DE UNA APLICACIÓN WEB PROGRESIVA PARA LA GESTIÓN DE PEDIDOS Y COMUNICACIÓN EN TIEMPO REAL DEL CAFÉ RESTAURANTE "PRODUCTOS CARLOS GERARDO" DEL TENA"**, propuesto por la estudiante Pico Jimenez Katherin Jamilep de la Carrera de Sistemas de Información, considero que dicho proyecto de titulación cumple con los requerimientos metodológicos y aportes científico-técnicos suficientes para ser sometidos al tribunal de lectores.

Ing. Victor Hugo Medina Matute, Mg.
C.C. 0501373955
TUTOR

Latacunga, marzo 2026

AVAL DE APROBACIÓN DE LECTORES

Cumpliendo con el Reglamento de Titulación de la Universidad Técnica de Cotopaxi, en calidad de Lectores de Tribunal de la Propuesta Tecnológica con el Título **"DESARROLLO DE UNA APLICACIÓN WEB PROGRESIVA PARA LA GESTIÓN DE PEDIDOS Y COMUNICACIÓN EN TIEMPO REAL DEL CAFÉ RESTAURANTE "PRODUCTOS CARLOS GERARDO" DEL TENA"**, propuesto por la estudiante Pico Jimenez Katherin Jamilep de la Carrera de Sistemas de Información, me permito indicar que la estudiante ha concluido todas las observaciones y realizado las correcciones señaladas por el Tribunal de Lectores, además de validar el funcionamiento de la propuesta, por lo cual presentamos el Aval de aprobación del Proyecto de Titulación correspondiente a la modalidad Propuesta Tecnológica en virtud de lo cual la postulante puede presentarse a la Defensa de su Proyecto de Titulación.

Particular que pongo en su conocimiento para los fines legales pertinentes.

Atentamente,

Lector 1 (Presidente)	Lector 2	Lector 3
Ing. Juan Chancusig	Ing. Karla Cantuña	Ing. Verónica Tapia
CC: 0502275779	CC: 0502305113	CC: 0502053697

AVAL DE IMPLEMENTACIÓN

Mediante el presente pongo a consideración que la estudiante **Katherin Jamilep Pico Jimenez**, realizó su tesis en beneficio del Café Restaurante **"Productos Carlos Gerardo"** del cantón Tena con el tema:

"Desarrollo de una Aplicación Web Progresiva para la Gestión de Pedidos y Comunicación en Tiempo Real del Café Restaurante 'Productos Carlos Gerardo' del Tena", trabajo que fue desarrollado con el propósito de mejorar la gestión de pedidos y optimizar la comunicación en tiempo real dentro del establecimiento, contribuyendo al fortalecimiento de los procesos de atención y servicio al cliente.

Se deja constancia de que la implementación del sistema se realizó de manera satisfactoria en beneficio del negocio mencionado.

PROCUTOS CARLOS GERARDO
RUC: 1500473606001
Telf: 0983310819
Tena - Napo - Ecuador

.....

Sr. Carlos Gerardo Morales Paredes
C.I. 1500473606
Propietario – Café Restaurante "Productos Carlos Gerardo"

AGRADECIMIENTO

Agradezco en primer lugar a Dios por brindarme la vida, la salud y la fortaleza necesaria para superar cada desafío y permitirme culminar esta importante etapa de mi formación académica.

A mis padres, Carmen Jiménez y Carlos Pico, por su amor, apoyo incondicional y sacrificio a lo largo de mi vida. Gracias por sus consejos, por creer siempre en mí y por motivarme a seguir adelante para alcanzar mis metas. A mis hermanos, Jeyson Pico y Carlos Pico, por su compañía, apoyo y por ser parte fundamental de mi vida durante este proceso.

A mi pareja, Angel Morales, por brindarme su apoyo incondicional, su paciencia y motivación en los momentos más difíciles, especialmente en aquellos días en los que sentí que quería rendirme.

A lo largo de todo este trayecto en la Universidad he conocido a muchas personas que han influido positivamente en mi vida, me han extendido su mano y me han brindado apoyo en cada etapa de este viaje. Quiero expresar un agradecimiento a cada una de ellas por aportar mucho en mi formación profesional, así como personal.

A mi tutor de tesis, el Ing. Víctor Medina, por su guía, conocimientos y acompañamiento durante el desarrollo de este trabajo de investigación.

Finalmente, expreso mi agradecimiento a la Universidad Técnica de Cotopaxi por haberme brindado la oportunidad de formarme académica y profesionalmente.

Katherin Jamilep Pico Jimenez

DEDICATORIA

Este trabajo está dedicado principalmente a Dios, por guiar mis pasos, darme fortaleza en los momentos difíciles y permitirme llegar hasta este punto tan importante de mi vida.

A mis padres, Carmen Jiménez y Carlos Pico, quienes con su amor, esfuerzo y ejemplo me han enseñado a luchar por mis sueños y a no rendirme ante las dificultades.

A mis hermanos, Jeyson Pico y Carlos Pico, por ser parte de mi vida y por compartir conmigo cada etapa de este camino.

A mi pareja, Angel Morales, por su amor, comprensión y por estar a mi lado brindándome ánimo y apoyo en cada momento.

Y de manera muy especial, a mi hija Angela Morales, quien es mi mayor inspiración y la razón que me impulsa cada día a seguir adelante y superarme. Este logro también es para ti, porque eres la luz que motiva cada uno de mis esfuerzos.

Este trabajo también está dedicado a todas aquellas personas que, de una u otra manera, aportaron con su granito de arena durante este proceso. Aunque no pueda mencionarlas a todas, siempre estaré profundamente agradecida por cada gesto de apoyo y por las buenas acciones que contribuyeron a la realización de este logro.

Con todo mi amor, dedico este logro a cada uno de ustedes, quienes forman parte fundamental de mi vida.

Katherin Jamilep Pico Jimenez

UNIVERSIDAD TÉCNICA DE COTOPAXI

FACULTAD DE CIENCIAS DE LA INGENIERÍA Y APLICADAS

TITULO: "DESARROLLO DE UNA APLICACIÓN WEB PROGRESIVA PARA LA GESTIÓN DE PEDIDOS Y COMUNICACIÓN EN TIEMPO REAL DEL CAFÉ RESTAURANTE "PRODUCTOS CARLOS GERARDO" DEL TENA"

Autor: Pico Jimenez Katherin Jamilep

RESUMEN

En el presente trabajo de titulación, bajo la modalidad de propuesta tecnológica tuvo como objetivo desarrollar una Aplicación Web Progresiva (PWA) para mejorar la gestión de pedidos y la comunicación en tiempo real en el Café Restaurante "Productos Carlos Gerardo" de la ciudad del Tena. El estudio surgió debido a que las actividades del restaurante, como la toma de pedidos, la comunicación entre el personal y el registro de cobros, se realizaban de manera manual, lo que ocasionaba retrasos en la atención, confusión en los pedidos y dificultades para mantener un control adecuado de la información. Para el desarrollo del proyecto se utilizó un enfoque de investigación mixto, combinando métodos cualitativos y cuantitativos. Se aplicó una entrevista al propietario y al personal del restaurante con el fin de conocer los procesos actuales y las necesidades del negocio. Además, se realizó una encuesta al personal para identificar su percepción sobre la utilidad de implementar un sistema web para mejorar las actividades diarias del establecimiento. El desarrollo de la aplicación se realizó mediante la metodología ágil eXtreme Programming (XP), la cual permitió organizar el proyecto en iteraciones cortas basadas en las fases de planificación, diseño, codificación y pruebas. La aplicación fue desarrollada utilizando el framework Django con el lenguaje de programación Python y una base de datos PostgreSQL. Asimismo, se implementó comunicación en tiempo real mediante WebSockets, lo que permitió que los pedidos registrados por los meseros se enviaran inmediatamente al área de cocina, facilitando la organización del trabajo y reduciendo posibles errores. Los resultados obtenidos evidenciaron que el personal del restaurante considera necesario implementar una herramienta tecnológica que permita optimizar la gestión de pedidos, mejorar la coordinación entre las áreas y brindar una atención más rápida a los clientes. Finalmente, se concluyó que la implementación de la aplicación web progresiva contribuye a mejorar la organización del trabajo, reducir los procesos manuales y fortalecer la gestión operativa del restaurante.

Palabras clave: Aplicación Web Progresiva, gestión de pedidos, comunicación en tiempo real, metodología XP, Django.

TECHNICAL UNIVERSITY OF COTOPAXI

FACULTY OF ENGINEERING AND APPLIED SCIENCES

TITLE: "DEVELOPMENT OF A PROGRESSIVE WEB APPLICATION FOR ORDER MANAGEMENT AND REAL-TIME COMMUNICATION OF THE CAFÉ RESTAURANT "PRODUCTOS CARLOS GERARDO" OF TENA"

Author: Pico Jimenez Katherin Jamilep

ABSTRACT

This degree project, developed under the technological proposal modality, aimed to design and implement a Progressive Web Application (PWA) to improve order management and real-time communication at the Café Restaurant "Productos Carlos Gerardo" in the city of Tena. The study emerged because several operational activities in the restaurant, such as order taking, communication among staff, and payment recording, were carried out manually. This situation generated delays in customer service, confusion in orders, and difficulties in maintaining an adequate control of information. To address this problem, a mixed research approach was applied, combining qualitative and quantitative methods. An interview was conducted with the restaurant owner and staff in order to identify the current operational processes and the main needs of the business. In addition, a survey was administered to the employees to determine their perceptions regarding the implementation of a web-based system to support daily activities within the establishment. The application was developed using the agile methodology eXtreme Programming (XP), which allowed the project to be organized into short development iterations that included planning, design, coding, and testing phases. The system was built using the Django framework, the Python programming language, and a PostgreSQL database. In addition, real-time communication was implemented through WebSockets, enabling orders registered by waiters to be instantly transmitted to the kitchen area, which improved workflow organization and reduced potential errors. The results indicated that the restaurant staff considers the implementation of a technological tool essential to optimize order management, improve coordination between work areas, and provide faster and more efficient customer service. Therefore, it was concluded that the implementation of the Progressive Web Application contributes to improving work organization, reducing manual processes, and strengthening the restaurant's operational management.

Keywords: Progressive Web Application, order management, real-time communication, XP methodology, Django.

UNIVERSIDAD TÉCNICA DE COTOPAXI

CENTRO DE IDIOMAS

AVAL DE TRADUCCIÓN

En calidad de Docente del Idioma Inglés del Centro de Idiomas de la Universidad Técnica de Cotopaxi; en forma legal **CERTIFICO** que:

La traducción del resumen al idioma Inglés de la Propuesta Tecnológica cuyo título versa: **"DESARROLLO DE UNA APLICACIÓN WEB PROGRESIVA PARA LA GESTIÓN DE PEDIDOS Y COMUNICACIÓN EN TIEMPO REAL DEL CAFÉ RESTAURANTE "PRODUCTOS CARLOS GERARDO" DEL TENA"**, presentado por la estudiante: **Katherin Jamilep Pico Jimenez**, de la carrera de **Sistemas de Información**, perteneciente a la **Facultad de Ciencias De La Ingeniería Y Aplicadas**, lo realizo bajo mi supervisión y cumple con una correcta estructura gramatical del Idioma.

Es todo cuanto puedo certificar en honor a la verdad por lo que autorizo a la peticionaria hacer uso del presente aval para los fines académicos legales.

Latacunga, marzo del 2026

Atentamente,

Mg. EDISON MARCELO PACHECO PRUNA.

DOCENTE DEL CENTRO DE IDIOMAS-UTC

CI: 0502617350

UNIVERSIDAD TÉCNICA DE COTOPAXI · LATACUNGA - ECUADOR · 24-01-1995

CENTRO DE IDIOMAS

ÍNDICE GENERAL

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

1. INFORMACIÓN GENERAL

Tema del proyecto:

Desarrollo de una Aplicación Web Progresiva para la Gestión de Pedidos y Comunicación en Tiempo Real del Café Restaurante “Productos Carlos Gerardo” del Tena

Modalidad de Titulación:

Tabla 1.1. Modalidad de titulación.

MODALIDAD DE TITULACIÓN	HOMOLOGACIONES PARA INFORME FINAL DE TITULACIÓN	SELECCIÓN
Propuesta tecnológica	Informe de propuesta tecnológica	X
	Patente, Modelo de utilidad, Certificado de propiedad intelectual.	
	Artículo científico	

Trabajo de Titulación Vinculado al Proyecto:

No vinculado

Equipo de Trabajo del Trabajo de Titulación:

Estudiante:

- Srta. Katherin Jamilep Pico Jimenez

Tutor de Titulación:

- Mgs. Victor Hugo Medina Matute

Área de Conocimiento:

Tabla 1.2. Área de conocimiento.

ÁREA CONOCIMIENTO	SUBÁREA CONOCIMIENTO	SUBÁREA ESPECIFICA CONOCIMIENTO
06 información y Comunicación (TIC)	061 información y Comunicación (TIC)	0611 el uso del Ordenador
		0612 base de datos, diseño y administración de redes
		0613 software y desarrollo y análisis de aplicativos

Línea de investigación: Tecnología de la información y las comunicaciones, robótica, automatización y optimización de sistemas.

Sublíneas de investigación de la Carrera: Ciencias informáticas para la modelación y automatización de sistemas a través de las TIC.

2. INTRODUCCIÓN

En la actualidad, el uso de la tecnología ha transformado la manera en que las organizaciones gestionan sus procesos internos. Especialmente en el sector gastronómico el cual ha experimentado una creciente digitalización de sus operaciones, impulsada por la necesidad de ofrecer un servicio más eficiente, personalizado y accesible. Los establecimientos que incorporan el uso de herramientas tecnológicas han logrado mejorado su atención al cliente mediante plataformas que permiten visualizar el menú, registros de pedido en línea y mantener una comunicación directa con el personal de trabajo.

Es por ello que, el Café Restaurante Productos Carlos Gerardo enfrenta dificultades debido a que la toma de pedidos y la comunicación interna se realizan de forma manual. Esta situación ha generado pérdida de información, retrasos en la atención, inconsistencias en la preparación de los alimentos y la falta del control administrativo sobre los pedidos y cobros, lo que genera desbalances en los ingresos del establecimiento.

Ante esta problemática, se propone el desarrollo de una Aplicación Web Progresiva (PWA) que atienda las necesidades del restaurante. Esta aplicación permitirá automatizar los procesos de gestión en una única plataforma la cual permitirá registrar pedidos, control de cobros, generar reportes y establecer una comunicación en tiempo real entre las áreas involucradas. Gracias a esta herramienta digital, los pedidos enviados por los meseros se visualizarán de forma

inmediata en la pantalla de cocina, lo que facilitará la preparación de los platos y reducirá errores operativos. Además, el administrador podrá consultar la información del negocio de manera rápida, segura y centralizada.

La elección de una PWA responde a sus características de accesibilidad y funcionamiento flexible, ya que puede funcionar por igual para todos es decir que independientemente del navegador web elegido y permite integrar elementos como notificaciones, sincronización inmediata y almacenamiento local [1]. Estas ventajas la convierten en una opción adecuada para negocios que requieren una solución ligera, escalable y compatible con distintos equipos.

Para el desarrollo de la aplicación se utilizará la metodología de Programación Extrema (XP), la cual se basa en generar entregables funcionales que implementa las historias de usuario asignadas a la iteración, este enfoque permite construir sistemas funcionales desde etapas tempranas y ajustar las funcionalidades conforme se identifican nuevas necesidades que el cliente requiera [2].

Al implementar la aplicación web, los trabajadores podrán gestionar los pedidos y realizar los cobros de forma segura y rápida. Por su parte, el administrador tendrá acceso a la generación de reportes y al control general de la aplicación. Esto reducirá el tiempo de toma de pedidos y la preparación de los platos, mejorando las operaciones internas del restaurante “Productos Carlos Gerardo”.

2.1 SITUACIÓN PROBLEMÁTICA

En América Latina se observa una diferencia significativa en el nivel de incorporación de las tecnologías de la información y comunicación (TIC), lo cual afecta especialmente a las micro y pequeñas empresas las cuales suelen contar con menos recursos y capacidades para adquirir herramientas tecnológicas. Lo que ha limitado su competitividad, la falta de infraestructura tecnológica y la implementación de sistemas que automaticen la gestión interna de las microempresas que están ubicadas en pequeñas ciudades [3].

Esto también se evidencia a nivel nacional en Ecuador, donde los establecimientos ubicados en ciudades pequeñas o sectores rurales aun manejan una gestión manual. Esta situación está relacionada con el hecho de que la mayoría de negocios en el país son microempresas, las cuales enfrentan limitaciones para automatizar sus procesos de gestión. Según el Registro Estadístico de Empresas (REEM) 2023 del INEC, el 93,67 % de las empresas ecuatorianas son

microempresas [4], lo que evidencia que aún continúa operando con recursos y capacidades limitadas para implementar sistemas digitales.

Además, esto se presenta en la provincia de Napo, en el cantón Tena, específicamente en el barrio Eloy Alfaro, donde el Café Restaurante "Productos Carlos Gerardo" gestiona sus operaciones de forma manual, incluyendo la toma de pedidos y los cobros. Esta práctica genera retrasos, duplicación de tareas y falta de control administrativo de los ingresos, afectando la calidad del servicio ofrecido. Asimismo, en el área de cocina se presentan demoras y confusiones al preparar los pedidos, ya que no existe una aplicación que organice y comunique claramente las órdenes de los clientes.

2.2 PREGUNATA DE INVESTIGACIÓN

¿Cuál es el nivel de usabilidad de la aplicación web progresiva desarrollada con Django para la gestión de pedidos y la comunicación interna en tiempo real en el restaurante "Productos Carlos Gerardo" de la ciudad del Tena?

2.3 OBJETO Y CAMPO DE ACCIÓN

2.3.1 Objeto de Investigación

Aplicación web progresiva para la gestión de pedidos y comunicación en tiempo real del Café Restaurante 'Productos Carlos Gerardo' del Tena

2.3.2 Campo de Acción

El campo de acción de este estudio se enmarca en el área **1203 Ciencia de los Ordenadores**, específicamente en las subáreas **1203.17 Informática**, **1203.18 Sistemas de Información y Diseño de Componentes** y **1203.23 Lenguajes de Programación**, considerando que el proyecto se centra en el desarrollo de una Aplicación Web Progresiva (PWA) para la gestión de pedidos y la comunicación en tiempo real

2.4 BENEFICIARIOS

2.4.1 Directo

Tabla 2.1. Beneficiarios directos.

	Cargo	**Descripción**	**N° de Personas**
Beneficiarios Directos	Dueño	Encargado de gestionar toda la información del aplicativo y supervisión de la aplicación	1
	Mesero	Responsable del control de pedidos.	4
	Cajero	Encargado de registrar y gestionar los cobros realizados.	3
	Cocinero	Encargado de recibir los pedidos en tiempo real, organizarlos y preparar los platos de forma rápida y ordenada	4
Beneficiarios directos totales			12

2.4.2 Indirecto

Tabla 2.2. Beneficiarios Indirectos.

Beneficiarios Indirectos	**Cargo**	**Descripción**	**N° de Personas**
	Clientes	Reciben atención más rápida, precisa y personalizada gracias a la aplicación.	250
Beneficiarios indirectos totales			250

2.5 JUSTIFICACIÓN

El desarrollo de esta propuesta tecnológica es relevante porque aborda la necesidad de optimizar la gestión y operación del Café Restaurante, mejorando la eficiencia, la precisión en los pedidos y la calidad del servicio ofrecido a los clientes. La implementación de soluciones digitales permite que el negocio se mantenga competitivo en un entorno cada vez más orientado a la tecnología y a la satisfacción inmediata del cliente.

El desarrollo de esta aplicación tiene un aporte práctico significativo, ya que permitirá automatizar la gestión de pedidos, cobros y la comunicación entre el personal. Con ello se reducirá el tiempo de atención y se asegurará que la información llegue de manera inmediata a cada área del restaurante. Esto favorece directamente el desempeño de los trabajadores del establecimiento quienes podrán realizar sus tareas de forma ordenada y correcta. Socialmente, se contribuye a fortalecer el emprendimiento local, apoyando a pequeños o grandes negocios en su proceso de transformación digital.

Desde el punto de vista teórico, este proyecto contribuye al conocimiento académico y científico al poner en práctica conceptos en programación, diseño de bases de datos, desarrollo de aplicaciones web y metodologías ágiles como XP. Asimismo, permite analizar como las Aplicaciones Web Progresivas (PWA) pueden mejorar procesos de negocio en entornos reales, generando evidencia sobre su eficacia, eficiencia y aplicabilidad. Esta experiencia aporta a la formación integral de los estudiantes, fortaleciendo la formación profesional y ampliando su capacidad para desarrollar soluciones tecnológicas basadas en fundamentos sólidos.

Los beneficiarios de este proyecto son distintos actores involucrados directa e indirectamente con la implementación del sistema. En primer lugar, el personal del restaurante se verá favorecido al contar con herramientas tecnológicas que faciliten la organización, el control y el seguimiento de las tareas diarias, lo que permitirá mejorar su desempeño y optimizar los procesos de trabajo. De igual manera, los clientes se beneficiarán al recibir un servicio más rápido, ordenado y confiable, ya que la automatización de los pedidos y la gestión interna del restaurante contribuirá a reducir tiempos de espera y posibles errores en la atención

Por otro lado, la viabilidad del proyecto se basa en varios elementos que aseguran su posibilidad de desarrollo e implementación. Desde la perspectiva técnica, hay herramientas y tecnologías adecuadas como el framework Django, el sistema gestor de bases de datos PostgreSQL y los estándares necesarios para el desarrollo de Aplicaciones Web Progresivas (PWA), que permiten crear un sistema efectivo, ampliable y adaptable a las exigencias del negocio. En el ámbito operativo, el restaurante posee el equipo y las condiciones requeridas para integrar el sistema y emplearlo en sus actividades cotidianas, garantizando su adecuado funcionamiento y mantenimiento. En términos económicos, la digitalización de los procedimientos hará posible disminuir gastos generados por errores humanos, pérdida de tiempo y gestión manual de datos, lo que convierte la inversión en el desarrollo del sistema en algo viable y lucrativo a mediano plazo.

2.6 OBJETIVOS

2.6.1 Objetivo General

Desarrollar una aplicación web progresiva mediante el Framework Django para automatizar la gestión de pedidos y la comunicación interna en tiempo real del restaurante “Productos Carlos Gerardo” de la ciudad del Tena.

2.6.2 Objetivo Específicos

- Analizar los fundamentos teóricos de la metodología XP y de la programación web progresiva (PWA) mediante revisión bibliográfica, con el fin de sustentar la fundamentación teórica del proyecto.
- Aplicar la metodología XP en el desarrollo de la aplicación web asegurando un proceso iterativo que permita construir un sistema funcional, escalable y adaptado a las necesidades de los usuarios.
- Implementar la aplicación web mediante pruebas de rendimiento y validación con usuarios para verificar la funcionalidad de la aplicación.

2.6.3 Sistema de tareas

Tabla 2.3. Sistema de tareas del plan de titulación.

Objetivos específicos	Actividades (tareas)	Resultados esperados	Técnicas, Medios e Instrumentos
Analizar los fundamentos teóricos de la metodología XP y de la programación web progresiva (PWA) mediante revisión bibliográfica, con el fin de sustentar la fundamentación teórica del proyecto.	• Búsqueda y selección de fuentes bibliográficas confiables. • Revisión documental de artículos, tesis y libros sobre la metodología XP. • Elaboración del marco teórico.	• Fundamento teórico claro y argumentado. • Identificación clara de los conceptos. • Base teórica suficiente para sustentar el proyecto.	• **Técnicas**: Revisión documental. • **Medios:** Artículos científicos, revistas, libros y gestor de referencia Mendeley. • **Instrumentos**: Ficha bibliográfica.
Aplicar la metodología XP en el desarrollo de la aplicación web asegurando un proceso iterativo que permita construir un sistema funcional, escalable y adaptado a las necesidades de los usuarios.	• Elaboración de historias de usuario. • Planificación de iteraciones de desarrollo. • Desarrollo incremental de funcionalidades del sistema.	• Historias de usuario. • Prototipos de interfaz del sistema. • Módulos del sistema desarrollados progresivamente mediante iteraciones.	• **Técnicas:** Entrevista y encuesta. • **Medios:** Reuniones con el propietario y personal del restaurante. • **Instrumentos:** Cuestionario de encuesta y guía de entrevista.

Objetivos específicos	Actividades (tareas)	Resultados esperados	Técnicas, Medios e Instrumentos
Implementar la aplicación web mediante pruebas de rendimiento y validación con usuarios para verificar la funcionalidad de la aplicación	• Desarrollo e integración de módulos del sistema. • Realización de pruebas internas y pruebas con usuarios. • Corrección de errores detectados.	• Aplicación web funcional y operativa. • Errores detectados y corregidos.	• **Técnicas:** Pruebas funcionales • **Medios:** Sistema implementado • **Instrumentos:** Casos de prueba.

3. FUNDAMENTACIÓN TEÓRICA

3.1 ANTECEDENTES

El desarrollo e implementación de sistemas de gestión para restaurante ha sido un campo recurrente de estudio en diversas universidades del Ecuador y Latinoamérica, dada la necesidad de mejorar los procesos operativos, administrativos y de atención al cliente en el sector gastronómico, especial en micro y pequeños establecimientos.

En la Universidad Técnica de Cotopaxi desarrollaron una tesis con el tema "Aplicación móvil para pedidos de comida a domicilio en la ciudad de Latacunga" utilizando tecnologías como MySQL, FireBase, Android Studio y PHP. El sistema desarrollado permitió gestionar de manera digital los pedidos de comida a domicilio provenientes de los restaurantes de la ciudad de Latacunga, esta solución ayudo a reducir la concentración de las personas en espacios cerrados con el fin de proteger la salud de los ciudadanos durante la pandemia. La metodología utilizada para el desarrollo de la aplicación es una adaptación de prácticas de Scrum que trabaja con el modelo Iterativo Incremental [5].

Por otro lado, en el Tecnológico Nacional de México se desarrolló una tesis con el tema "Sistema web para la automatización del servicio en restaurantes" utilizando tecnologías como MySQL, JavaScript y HTML. Este sistema web permitió la automatización de la gestión de procesos y servicios en restaurantes del municipio de Cuautitlán Izcalli disminuye el riesgo en posibles fallas que se pueden presentar durante la atención. La metodología utilizada para el desarrollo de esta tesis es XP ya que está diseñada para entregar el software que los clientes necesitan en el momento en que lo necesitan [6].

En la Universidad Regional Autónoma de Los Andes "UNIANDES" se desarrolló una tesis para la carrera de sistemas con el tema "Aplicación web para la gestión del servicio al cliente en el restaurante Innovation Food de la ciudad de Ambato". La implementación de la aplicación Web reducirá los tiempos de atención a los clientes, el mismo que además dispondrá de la información oportuna y completa de los productos que ofrece el restaurante con esto se tendrán clientes satisfechos y mayor ingreso por volumen de ventas [7].

3.2 BASES TEÓRICAS

3.2.1 Aplicaciones web

Una aplicación web es aquella que reside en un ordenador, denominado servidor web y que los usuarios pueden utilizar a través de internet mediante un navegador web para obtener los servicios que ofrezca dicha aplicación.

Existen diversos tipos de aplicaciones web, tales como gestores de correo web, wikis, blogs, tiendas en línea, entre otros tipos; además, según su tipo de acceso, las aplicaciones web pueden ser:

- **Públicas:** Como tiendas virtuales, diarios digitales, portales de internet, entre otras.
- **Restringidas:** Como las intranets, las cuales ofrecen servicios para mejorar las gestiones internas en empresas, tales como gestión de proyectos y tareas, control de horas de trabajo, gestores de documentos.

Además, existen las páginas web estáticas las cuales solo muestran información al usuario de forma limitada, ya que no permiten interactuar con la página web. Estas páginas están construidas principalmente con hipervínculos o enlaces a otras páginas web. Algunos ejemplos pueden ser foros, consultas online, redes sociales, entre otras.

Y las páginas web dinámicas son aquellas que contienen elementos que permiten una interacción activa entre el usuario y la aplicación. Estas aplicaciones permiten que el usuario visualice e ingrese datos de modo interactivo, en donde la página web responde a cada una de sus acciones, por ejemplo, consultar el correo, rellenar y enviar formularios[8].

3.2.1.1 Aplicación Web Progresiva

Una aplicación web progresiva busca integrar y aprovechar las ventajas de las aplicaciones nativas junto con el alcance que ofrecen las aplicaciones web, estas aplicaciones ofrecen funcionamiento sin conexión, instalación en el dispositivo, notificaciones push y acceso desde un icono. Lo que permite que los usuarios interactúen con la aplicación de manera eficiente y fluida [9].

3.2.1.1.1 Arquitectura de una PWA

La PWA se basa en la combinación de una aplicación web tradicional con tecnologías que permiten mejorar su funcionamiento y experiencia de uso. De forma general es la integración

de una aplicación web bajo el protocolo HTTPS y que utiliza un Service Worker para gestionar ciertos procesos en segundo plano.

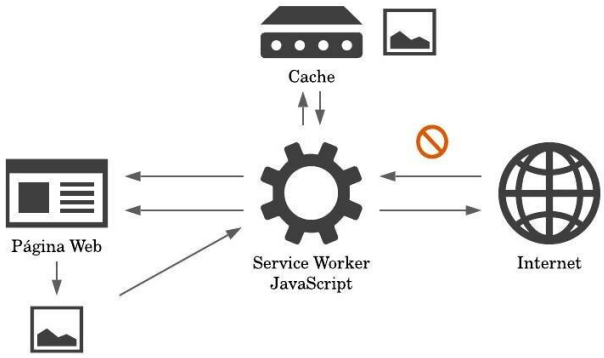


Figura 3.1. Arquitectura de una PWA[10].

3.2.1.1.2 Características de una PWA

Instalación: Se puede acceder a la aplicación mediante un navegador web y existe la posibilidad de "instalar" en la pantalla de inicio del dispositivo, sin necesidad de descargarla desde una tienda de aplicaciones.

Multiplataforma: Estas aplicaciones son compatibles con múltiples plataformas ya que pueden funcionar en cualquier dispositivo que disponga de un navegador web sin depender del sistema operativo.

Actualizaciones: Las actualizaciones se realizan automáticamente cuando el usuario ingresa nuevamente a la aplicación desde el navegador asegurando así que los usuarios utilicen su última versión.

Acceso sin conexión: Algunas aplicaciones web progresivas pueden utilizarse incluso sin conexión a internet gracias al almacenamiento en caché de la información previamente.

Distribución: Pueden compartirse de forma sencilla mediante un enlace web, sin requerir procesos de publicación o revisión en tiendas de aplicaciones [11]

3.2.1.1.3 Diferencias de una PWA frente a una aplicación web tradicional y nativa

Las aplicaciones web convencionales son páginas en Internet que proporcionan interactividad y operan completamente a través del navegador. Necesitan una conexión a la red, no se descargan en el dispositivo, y su acceso a las funciones del hardware es limitado. Su mayor beneficio es que pueden operar en cualquier dispositivo que tenga un navegador, eliminando la necesidad de crear distintas versiones para cada sistema operativo.

Las aplicaciones nativas son creadas específicamente para un sistema operativo particular (como iOS, Android o Windows). Proporcionan un rendimiento óptimo y acceso total al hardware del dispositivo, pero requieren que se instalen a través de tiendas de aplicaciones, ocupan espacio en la memoria y demandan actualizaciones manuales. Además, diseñar aplicaciones nativas para diferentes plataformas significa que hay que programar de manera distinta para cada una, lo que incrementa tanto el tiempo como los costos de desarrollo.

Las aplicaciones web progresivas integran características de ambos tipos. Al igual que las aplicaciones web, emplean estándares de Internet y son accesibles en varias plataformas desde un único código base. De manera similar a las aplicaciones nativas, pueden ser instaladas en el dispositivo, operar sin conexión, enviar notificaciones y acceder a algunas funciones del hardware. Su desarrollo es más ágil y económico en comparación con la creación de varias aplicaciones nativas, al tiempo que brindan una experiencia mejorada que las aplicaciones web tradicionales.

3.2.1.1.4 Ventajas de utilizar una PWA en la gestión de pedidos.

El uso de aplicaciones web progresivas ofrece diversos beneficios tanto para los usuarios como para los desarrolladores. Entre ellos se destaca la facilidad de acceso a la aplicación, ya que los usuarios pueden utilizarla e instalarla rápidamente sin depender de tiendas de aplicaciones. Asimismo, permiten mantener la disponibilidad del servicio incluso cuando la conexión a Internet es limitada, lo que mejora la experiencia de uso. También facilitan la gestión y actualización del sistema, puesto que los cambios se aplican automáticamente sin requerir acciones por parte del usuario. Otro beneficio importante es la optimización del uso de recursos del dispositivo, debido a que estas aplicaciones suelen requerir menos espacio de almacenamiento y menor consumo de datos. Desde el punto de vista del desarrollo, las PWA reducen tiempos y costos de implementación, al permitir trabajar con una sola base de código compatible con diferentes plataformas. Finalmente, contribuyen a ofrecer una experiencia de uso ágil y segura, favoreciendo tiempos de carga rápidos y el uso de conexiones protegidas [12].

Considerando lo previamente mencionado en relación al sistema creado para la administración de pedidos en el restaurante, la implementación de una PWA permite que el equipo acceda al sistema desde una variedad de dispositivos como teléfonos inteligentes, tabletas o computadoras sin la necesidad de descargar la app desde una tienda. Esto simplifica su utilización dentro del ámbito laboral del restaurante, donde los camareros, cajeros, clientes y el

personal de cocina necesitan ingresar al sistema rápidamente desde múltiples dispositivos. Estas características convierten a las PWA en una opción eficaz frente a las aplicaciones web convencionales, particularmente en sistemas que demandan accesibilidad, velocidad y sencillez de uso en diversas plataformas.

Por estas razones, se optó por implementar el sistema con una aplicación web progresiva, ya que mejora la accesibilidad del sistema, optimiza la experiencia del usuario y facilita su uso en dispositivos móviles en el entorno del restaurante.

3.2.1.1.5 Herramientas para desarrollar una PWA

3.2.1.1.5.1 Framework Django

Django es un framework web de alto nivel basado en Python que facilita el desarrollo rápido y eficiente de aplicaciones web. Se trata de una herramienta gratuita y de código abierto que cuenta con una amplia documentación y soporte gratuito. Las aplicaciones desarrolladas con Django suelen organizarse en diferentes archivos, cada uno encargado de gestionar funciones específicas del sistema.

Vista (View): Las vistas son funciones de Python responsables de procesar las solicitudes HTTP y devuelven respuestas HTTP. Las vistas interactúan con los modelos para obtener la información necesaria y utilizar las plantillas para presentar los resultados al usuario.

Plantillas (Templates): Una plantilla es un archivo de texto que define la estructura de otro documento por ejemplo una página HTML. Dentro de este archivo se incluyen marcadores de posición que posteriormente se reemplazan con contenido dinámico cuando se genera la página final [13].

URLs: Donde se definen los patrones de URL para tu aplicación. Estos patrones de URL determinan cómo se asignan las solicitudes a las vistas.

Modelos (Models): Los modelos especifican la estructura de la base de datos que utiliza tu aplicación. El ORM que viene con Django facilita enormemente la gestión de bases de datos [14].

3.2.1.1.5.2 Django Channels

Django Channels es un proyecto que te permite manejar WebSockets, protocolos de chat, protocolos de IoT y más. Es construido sobre una especificación de Python llamada ASGI. Channels se basa en el soporte ASGI nativo en Django. Mientras Django todavía maneja HTTP

tradicional, los canales le brindan la opción de manejar otras conexiones en ya sea un estilo sincrónico o asincrónico [15].

3.2.1.1.5.3 Websocket

WebSocket es un protocolo de comunicación que permite que una aplicación cliente y una aplicación de servidor web utilicen una conexión dúplex, es decir, que pueden enviar y recibir datos al mismo tiempo, haciendo así que la transmisión de información sea en tiempo real [16].

3.2.1.1.5.4 Python

Python es un lenguaje de programación interpretado de alto nivel y de código abierto lo que permite su uso sin costo en distintos sistemas operativos como Windows, Linux, entre otros, se caracteriza por tener una sintaxis clara y sencilla, el cual permite el desarrollo rápido de sistemas de información web; además, se caracteriza por ser muy adecuado para el proceso de enseñanza y aprendizaje [17].

Figura 3.2. Python [18].

3.2.1.1.5.5 HTML5

HTML5 corresponde a las siglas de *HyperText Markup Language* o Lenguaje de Marcado de Hipertexto. Este lenguaje se utiliza para estructurar el contenido de las páginas web mediante etiquetas que permiten definir elementos como textos, imágenes y otros recursos. Estas etiquetas son interpretadas por los navegadores web, el cual mostrará de manera adecuada la información en la página web. Además, HTML5 no se limita solo a ser un lenguaje de marcado que solo permite definir elementos básicos de una página, ya puede integrarse con otros lenguajes como CSS y JavaScript, que permiten mejorar la apariencia visual y añadir funcionalidades interactivas a los sitios web [19].

Figura 3.3. HTML5 [20].

3.2.1.1.5.6 CSS

CSS (*Cascading Style Sheets*) es un lenguaje de hojas de estilo utilizado para definir la apariencia y el diseño visual de las páginas web. Las hojas de estilo CSS fueron creadas para separar el contenido del documento de su presentación gráfica. Este lenguaje cuenta con su propia sintaxis y permite manejar selectores, comentarios y distintos elementos de estilo. CSS se utiliza para mejorar el diseño de las etiquetas HTML5 como tipografías, colores, márgenes, bordes , altura, ancho, animación y entre otros [21].

Figura 3.4. CSS3 [22].

3.2.1.1.5.7 JavaScript

Habitualmente el código JavaScript se integra dentro de documentos HTML y es interpretado por el navegador cuando se visualiza la página web. Su principal función es permitir la validación de datos ingresados por los usuarios y la creación de contenidos dinámicos dentro de las páginas HTML [23].

Figura 3.5. JavaScript [24].

3.2.1.1.5.8 AJAX

AJAX (Asynchronous JavaScript and XML) es una colección de clases especiales basadas en scripts. La cual permite que el cliente y el servidor interactúen de forma asíncrona es decir que solo una parte de la página se recrea, se reformatea, se devuelve al cliente y se vuelve a renderizar en el navegador del cliente, se produce una renderización parcial de la página [25].

3.2.1.1.5.9 Bootstrap

Bootstrap es un framework de código abierto utilizado para el diseño y desarrollo de sitios web adaptables (*responsive*) a través del uso de sus librerías CSS Y JavaScript. Gracias a sus componentes como menús, formularios, botones, cuadros, ventanas modales y otros elementos, permite diseñar páginas web que se ajustan a diferentes dispositivos y necesidades de los usuarios [26].

Figura 3.6. Bootstrap [27].

3.2.1.1.5.10 Visual Studio Code

Visual Studio Code es un editor de código para programadores gratuito, de código abierto y multiplataforma. Ofrece herramientas de apoyo al desarrollo como depuración, control de versiones, resaltado de sintaxis y extensiones; además de eso, VS Code tiene soporte integrado para múltiples lenguajes de programación. Este soporte se extiende a más de 30 lenguajes a

través de sus extensiones, lo que hace que sea una herramienta versátil para cualquier proyecto [28].

Figura 3.7. Visual Studio Code [29].

3.2.2 Gestor de base de datos

Los Sistemas Gestores de Bases de Datos (SGBD), conocidos en inglés como *Database Management System* (DBMS), son herramientas que permiten crear, administrar y organizar las bases de datos. Estos sistemas facilitan el almacenamiento, la gestión y la recuperación de la información mediante diversas estructuras de datos, y se pueden clasificarse en dos grupos los relacionales (SQL), que organizan los datos en tablas relacionadas entre sí y los no relacionales (NoSQL), que no requieren estructuras rígidas y suelen emplearse en entornos distribuidos.

3.2.2.1 Sistemas gestores de bases de datos relacionales (SQL)

Los principales sistemas gestores de bases de datos relacionales (SGBD SQL) más utilizados actualmente son:

3.2.2.1.1 MySQL

Es un sistema de gestión de bases de datos que permite múltiples hilos y usuarios, ampliamente empleado en numerosas páginas de internet hoy en día, siendo el más frecuente en aplicaciones de código abierto. Las ventajas más destacadas de este gestor de base de datos son:

- Simples de utilizar y alto rendimiento.
- Facilidad para instalar y configurar.
- Soporte multiplataforma.
- Certificación SSL.

Una desventaja notable es su limitación en escalabilidad, ya que no maneja eficazmente bases de datos de gran tamaño.

3.2.2.1.2 Maria DB

El sistema de gestión de bases de datos MariaDB es una variante de MySQL que retiene la mayor parte de las funcionalidades de su predecesor y añade diversas mejoras. Surge como consecuencia de la compra de MySQL por Oracle, manteniendo el enfoque de código abierto y ofreciendo total compatibilidad con MySQL.

Entre las características más destacadas de este gestor de bases de datos se incluyen:

- Incremento de opciones de motores de almacenamiento.
- Alta capacidad de escalabilidad.
- Eficiencia y seguridad en las transacciones.
- Mejoras y nuevas funcionalidades enfocadas a su uso en bases de datos NoSQL.

Las desventajas son mínimas, destacando solo algunas incompatibilidades menores durante la migración entre MariaDB y MySQL, así como ligeros retrasos en la publicación de versiones estables.

3.2.2.1.3 Microsoft SQL Server

El sistema de gestión de bases de datos Microsoft SQL Server está diseñado para ser un gestor de bases de datos relacionales utilizando el lenguaje Transact-SQL, permitiendo a múltiples usuarios acceder a grandes volúmenes de datos al mismo tiempo.

Este software es de propiedad exclusiva de Microsoft. Sus características más destacadas incluyen:

- Soporte único por parte de Microsoft.
- Capacidad de escalabilidad, estabilidad y seguridad.
- Opción de cancelar consultas en curso.
- Un poderoso entorno gráfico para la administración que permite manejar comandos DDL y DML.
- Aunque fue originalmente creado para Windows, ha estado disponible en otras plataformas como Linux o Docker por algún tiempo.

Su principal inconveniente es el costo. Ofrece una versión gratuita (Express), pero lo habitual es optar por alguna de las variantes de pago (Standard, Developer, Enterprise o SQL Azure, que es la edición de SQL Server en la nube).

3.2.2.1.4 PostgreSQL

PostgreSQL es un sistema gestor de bases de datos relacional orientado a objetos, de código abierto y distribuido bajo la licencia BSD. Se caracteriza por su alto rendimiento y estabilidad, lo que lo convierte en una opción adecuada para aplicaciones que requieren un manejo eficiente y seguro de la información.

Entre sus principales características se encuentran:

- Control de concurrencia multiversión (MVCC).
- Compatibilidad con diversos lenguajes de programación.
- Multiplataforma.
- Disponibilidad de la herramienta pgAdmin, que facilita la administración y gestión de las bases de datos mediante una interfaz intuitiva.

3.2.2.2 Sistemas gestores de bases de datos No Relacionales (NoSQL)

Los principales sistemas gestores de bases de datos no relacionales más usadas son:

3.2.2.2.1 MongoDB

Estamos hablando del sistema de gestión de bases de datos NoSQL que es el más conocido y usado en la actualidad.

MongoDB es un sistema de gestión de bases de datos NoSQL que se enfoca en documentos y guarda datos en formatos BSON, ofreciendo un esquema flexible que facilita su integración.

Negocios como Google, Facebook, eBay, Cisco y Adobe emplean MongoDB como su solución de gestión de bases de datos.

Las características más destacadas de MongoDB incluyen:

- Indexación y replicación.
- Distribución de carga.
- Almacenamiento basado en documentos.
- Consultas específicas.

- Escalabilidad horizontal.
- Código abierto.

Una desventaja notable de MongoDB es que no es la opción ideal para manejar transacciones complejas.

3.2.2.2.2 Redis

Se fundamenta en el almacenamiento por pares clave-valor. Podemos imaginar el sistema de gestión de bases de datos Redis como un gran contenedor que guarda diversos tipos de información, como cadenas, hashes, listas, entre otros.

La utilización principal de este sistema de gestión de bases de datos es la gestión en memoria caché y el control de sesiones.

Sus características más destacadas son:

- Atomicidad y conservación de datos.
- Alta velocidad.
- Facilidad de uso.
- Compatibilidad con múltiples plataformas.

3.2.2.2.3 Cassandra

De manera similar a Redis, Cassandra, el sistema de gestión de bases de datos, también se basa en el almacenamiento de tipo clave-valor. Se trata de un sistema de bases de datos NoSQL que es tanto distribuido como enormemente escalable.

Plataformas como Facebook, Twitter, Instagram, Spotify y Netflix hacen uso de Cassandra.

Cuenta con un lenguaje específico para realizar consultas, conocido como CQL (Cassandra Query Language).

Las características principales de este sistema de gestión de bases de datos NoSQL son:

- Compatibilidad con múltiples plataformas.
- Lenguaje de consulta propio (CQL).
- Escalabilidad tanto lineal como horizontal.
- Es un sistema de bases de datos distribuido.

- Emplea una arquitectura de tipo peer-to-peer [30].

3.2.3 Metodología de desarrollo del software

3.2.3.1 Metodologías ágiles

Las metodologías ágiles representan un enfoque de desarrollo iterativo e incremental, es decir, se realizan ciclos de desarrollo cortos y repetitivos que entregan incrementos funcionales del producto y en cada uno de estos ciclos se llevan a cabo actividades que se realizan en todas las fases de la metodología, empezando por la recolección de requerimientos, prototipos, verificaciones y entregas; estos ciclos cortos son conocidos como sprints. Este tipo de metodologías promueven la entrega continua de software funcional, permitiendo así que el sistema evolucione de manera incremental incorporando nuevas funcionalidades. La metodología ágil se utiliza para el desarrollo de aplicaciones o de otros tipos de software, ya que cambia constantemente para así llegar a cumplir con los requerimientos del usuario; además, esta metodología favorece al trabajo en equipo y la comunicación constante entre los miembros del equipo de desarrollo.

3.2.3.2 Principios básicos del Agile Manifesto

El Manifiesto Ágil es un texto que destaca 4 valores y 12 principios para la creación de software utilizando enfoques ágiles. Fue dado a conocer en febrero de 2001 por un grupo de 17 programadores que buscaban una opción diferente al método de desarrollo de productos que era más rígido y secuencial.

3.2.3.2.1 Los 4 valores de las metodologías ágiles

Según lo expuesto en el Manifiesto Ágil, existen cuatro principios fundamentales en la gestión ágil de proyectos:

1. **Personas e interacciones sobre procesos y herramientas:** En las metodologías ágiles se prioriza la comunicación y colaboración entre los miembros del equipo por encima del uso rígido de procesos o herramientas. Este enfoque promueve el trabajo conjunto y el apoyo mutuo para alcanzar mejores resultados en el desarrollo del proyecto.
2. **Software funcionando sobre documentación extensa:** El desarrollo ágil da mayor importancia a la entrega de software funcional que a la elaboración de documentación excesiva. El objetivo principal es proporcionar al usuario un producto que funcione correctamente y satisfaga sus necesidades.

3. **Colaboración con el cliente sobre negociación contractual:** Las metodologías ágiles fomentan la interacción constante con el cliente, permitiendo que este participe en la orientación del desarrollo del software. Por ello, la cooperación continua se considera más importante que los detalles formales de los contratos.
4. **Respuesta al cambio sobre seguir un plan:** El enfoque ágil destaca la capacidad de adaptación frente a los cambios que puedan surgir durante el proyecto. Esto permite ajustar estrategias y decisiones sin afectar negativamente el desarrollo del producto.

3.2.3.2.2 Los 12 principios ágiles del Manifiesto Ágil

Los cuatro valores del Manifiesto Ágil constituyen la base del enfoque ágil. A partir de ellos se establecen doce principios que orientan las prácticas de desarrollo y pueden adaptarse a las necesidades de cada equipo.

1. **Satisfacer a los clientes con entrega temprana y continua:** Si los clientes reciben actualizaciones periódicas, es más probable que vean los cambios deseados en el producto y la satisfacción del cliente sea positiva. Las consecuencias son que los clientes se vuelven más felices y satisfechos; Además, hay más colecciones repetidas. Deleitar al cliente es clave en el Manifiesto Ágil.
2. **Los requisitos pueden modificarse incluso en fases avanzadas del proyecto:** Uno de los pilares de las metodologías ágiles es la capacidad de adaptación. En los procesos iterativos, esta flexibilidad permite responder mejor a los cambios y aprovecharlos en beneficio del desarrollo.
3. **Se realizan entregas frecuentes de valor para el cliente:** Al proporcionar resultados útiles de manera constante, se fortalece la relación con los usuarios y se reduce la posibilidad de perder su confianza o interés.
4. **Se promueve la colaboración dentro del equipo:** La estrategia ágil busca eliminar el trabajo aislado y fomentar que los miembros del proyecto trabajen de forma conjunta, compartiendo responsabilidades y conocimientos.
5. **Los proyectos se desarrollan con equipos motivados:** La metodología ágil obtiene mejores resultados cuando las personas se sienten comprometidas con sus tareas y trabajan activamente para cumplir los objetivos del proyecto, favoreciendo así la calidad técnica.

6. **La comunicación directa es la forma más efectiva de intercambio de información:** Aunque las dinámicas laborales han evolucionado con el tiempo, la interacción cara a cara sigue siendo fundamental; en equipos distribuidos, herramientas como videollamadas ayudan a mantener una comunicación cercana.
7. **El progreso del proyecto se mide principalmente por el funcionamiento del software:** En las metodologías ágiles se prioriza la entrega de productos operativos, ya que el software funcional es el indicador más claro de avance.
8. **Se busca mantener un ritmo de trabajo sostenible:** Aunque los métodos ágiles pueden implicar rapidez en ciertos momentos, es importante evitar la sobrecarga del equipo y mantener una constancia que permita avanzar sin generar agotamiento.
9. **La mejora continua y la calidad técnica fortalecen la agilidad del desarrollo** Cuando se produce código de calidad en cada iteración, este puede servir de base para futuros avances, facilitando un desarrollo más eficiente y sostenible.
10. **La simplicidad es un principio clave:** En muchos casos, las soluciones más sencillas son las más efectivas, por lo que las metodologías ágiles buscan evitar la complejidad innecesaria y enfocarse en resolver los problemas de forma clara y práctica.
11. **Los equipos autoorganizados generan mejores resultados:** Cuando los miembros del equipo trabajan con autonomía y proactividad, logran crear productos y soluciones que aportan mayor valor al proyecto y a la organización.
12. **El equipo evalúa constantemente su desempeño para mejorar**: A través de reuniones de retrospectiva, los integrantes analizan lo realizado, identifican oportunidades de mejora y ajustan sus métodos de trabajo para aumentar la eficacia en las siguientes etapas.

3.2.3.3 Tipos de metodologías ágiles

Existen distintos tipos de metodologías agiles las cuales son:

3.2.3.3.1 Kanban

Kanban proporciona un enfoque visual para aplicar las prácticas ágiles. Los equipos suelen utilizar herramientas en línea con tableros Kanban, como Trello y Asana, para representar las diferentes tareas dentro del proceso de desarrollo. Estas tareas se muestran en forma de tarjetas dentro de un tablero, mientras que las etapas del proceso se organizan en columnas. A medida

que los miembros del equipo avanzan en su trabajo, las tarjetas se trasladan desde la columna de tareas pendientes hacia otras columnas que indican la etapa en la que se encuentra cada actividad.

Este método permite a los equipos identificar posibles obstáculos y visualizar con mayor claridad la cantidad de trabajo que se está realizando.

3.2.3.3.2 Scrum

La metodología Scrum es ampliamente utilizada para implementar prácticas ágiles en equipos pequeños. Dentro de este enfoque existe la figura del Scrum Master, quien se encarga de coordinar al equipo y eliminar los obstáculos que puedan dificultar el trabajo de los demás integrantes. Las actividades se desarrollan en periodos llamados sprints, y los equipos que aplican Scrum realizan reuniones diarias para revisar las tareas en curso, detectar problemas y analizar cualquier aspecto que pueda influir en el avance del proyecto[31].

3.2.3.3.3 Metodología XP

La Programación Extrema (XP) es una metodología ágil la cual se centra en velocidad, simplicidad y calidad del desarrollo del software, mediante ciclos de desarrollo cortos y repetitivos. Esta metodología cuenta con menos carga de documentación, lo cual permite enfocarse más en el desarrollo del sistema para así llegar a cumplir con todos los requerimientos del usuario.

Esta metodología es un enfoque de desarrollo de software dividido en ciclos de trabajo cortos, en el que al finalizar cada uno se revisa y evalúa el incremento funcional del sistema, permitiendo su refinamiento para adaptarlo a los requisitos y alcanzar la eficiencia máxima. La programación extrema es una metodología disciplinada, ya que realiza revisiones de código frecuentes y pruebas para realizar cambios rápidamente. Además, fomenta la creatividad y el trabajo colaborativo, impulsando la participación del equipo de desarrollo durante todas las etapas del proceso, lo que contribuye a mejorar la calidad del software y la satisfacción del usuario final.

3.2.3.3.3.1 Fases de la metodología XP

Dentro de la metodología XP existe diferentes fases las cuales se realizan dentro de cada iteración las cuales se describen a continuación:

- **Planificación:** La planificación es la fase inicial donde se determina en la cual se analiza la viabilidad del proyecto y su compatibilidad con la metodología XP para evaluar esto se realizan las historias de usuario donde se va a confirma la complejidad del proyecto.
- **Diseño:** En esta fase se diseñan los templates del sistema siguiendo el orden establecido en las iteraciones de desarrollo, comenzando por las interfaces más sencillas hacia las más complejas del proyecto.
- **Codificación:** La codificación es el proceso mediante el cual se desarrolla el código fuente del sistema, y en donde se realiza de forma paralela con el diseño, en el cual se implementan las funcionalidades definidas utilizando los lenguajes de programación seleccionados.
- **Prueba:** Esta es la última fase de la metodología XP, en donde se realizan pruebas unitarias durante todo el proceso de programación [32].

3.2.3.3.3.2 Roles de la metodología XP

En la metodología XP se definen varios roles, los cuales permiten organizar el trabajo del equipo de desarrollo, lo cual promueve la comunicación constante, la colaboración y la entrega continua de módulos funcionales. Cada rol cumple una función específica dentro del proceso de desarrollo, asegurando que los requerimientos del cliente sean correctamente interpretados.

Estos roles no son rígidos, ya que una misma persona puede desempeñar más de un rol dependiendo del tamaño del proyecto, aunque su correcta definición mejora la calidad del producto final.

A continuación, se describen los principales roles que intervienen en la metodología eXtreme Programming:

- **Programador:** El programador es el responsable de diseñar, desarrollar e implementar las funcionalidades del sistema, es decir, que es quien produce el código; además, participa activamente en el análisis de los requerimientos y en la construcción de las historias de usuario.
- **Cliente:** El cliente representa a los usuarios finales del sistema y es el encargado de definir las historias de usuario para obtener así los requerimientos del proyecto; además, es quien validará el funcionamiento e implementación del sistema [33].

- **Encargado de pruebas (Tester):** El tester es el encargado de las pruebas y es quien verifica el correcto funcionamiento del sistema mediante la ejecución del sistema para así validar que las funcionalidades implementadas cumplan con los requerimientos establecidos, así como en documentar los resultados de las pruebas realizadas para garantizar la calidad del software.
- **Encargado de seguimiento (Tracker):** El tracker es el encargado de dar seguimiento al progreso del proyecto, evaluando la relación entre el tiempo estimado para las actividades y el tiempo real utilizado en su ejecución, comunicando los resultados para mejoras futuras. Este rol evalúa el desempeño del equipo y proporciona información objetiva sobre el estado del proyecto durante las iteraciones.
- **Entrenador (Coach):** El coach es responsable de guiar al equipo en la correcta aplicación de la metodología eXtreme Programming. Este rol apoya al equipo de desarrollo en la adopción de las prácticas XP [34].

3.3 GESTIÓN DE PEDIDOS Y TRANSFORMACIÓN DIGITAL EN RESTAURANTES

La transformación digital en los restaurantes facilita la gestión de las actividades empresariales, incrementando la productividad de las empresas para así lograr ventajas competitivas mediante el ahorro de tiempo y costo, además de permitir una mejor calidad del servicio, la toma de decisiones y la captación de nuevas oportunidades de mercado. Aunque los procesos de innovación se ven obstaculizados debido al costo, el riesgo asociado y la necesidad de contar con servidores actuales [35].

Desde una perspectiva operativa, la automatización de la gestión de pedidos reduce los costos laborales, minimiza los errores humanos y acelera la prestación del servicio, además la capacidad de recopilar y analizar datos de los clientes permite a los restaurantes refinar la oferta de menús e implementar estrategias comerciales más efectivas. Por lo que un sistema de pedidos digitales que reduzca las imprecisiones en los pedidos y el largo tiempo de espera, permitiendo así a los clientes realizar y personalizar su pedido, y la función de pago garantiza una experiencia gastronómica más segura por lo que optimiza la gestión y protección de los datos confidenciales de los clientes [36].

3.4 RESTAURANTE

El Café Restaurante “Productos Carlos Gerardo” se encuentra ubicado en la provincia de Napo, en el cantón Tena, específicamente en el barrio Eloy Alfaro, frente al Hospital General José María Velasco Ibarra.

Este establecimiento se dedica a la preparación y venta de alimentos y bebidas, ofreciendo a sus clientes un espacio cómodo y acogedor donde pueden disfrutar de desayunos, almuerzos, meriendas y café. El restaurante busca brindar una atención cordial y un servicio de calidad, atendiendo tanto a los habitantes del sector como a visitantes que transitan por la zona.

Además, su ubicación estratégica frente al hospital lo convierte en un punto de fácil acceso para pacientes, familiares, personal médico y público en general que necesitan un lugar cercano para alimentarse o descansar mientras realizan sus actividades. El Café Restaurante “Productos Carlos Gerardo” se caracteriza por ofrecer productos preparados con ingredientes frescos y un ambiente familiar, contribuyendo así a la oferta gastronómica local del cantón Tena.

De esta manera, el establecimiento cumple un papel importante dentro de la comunidad, ya que no solo brinda servicios de alimentación, sino que también genera un espacio de encuentro para las personas del sector, fortaleciendo la economía local y apoyando el desarrollo de pequeños emprendimientos gastronómicos en la región [37].

Figura 3.8. Restaurante "Productos Carlos Gerardo".

4. MÉTODOS Y PROCEDIMIENTOS

4.1 ENFOQUE DE INVESTIGACIÓN

Para el desarrollo de este proyecto se adoptó un enfoque de investigación mixto ya que al integrar métodos cualitativos y cuantitativos para la recolección, análisis e interpretación de la información obtenida. Este enfoque permite aprovechar las fortalezas de ambos métodos proporcionando mejores resultados completos y confiables.

El enfoque cualitativo se centra en la recolección y análisis de datos no numéricos, el cual se aplicó mediante una entrevista semiestructurada la cual permitió comprender a profundidad los procesos operativos actuales del establecimiento identificando así la problemática existente, requerimientos funcionales y necesidades de los trabajadores.

El enfoque cuantitativo se centró en la recopilación y análisis de datos numéricos para poder identificar patrones y tendencias, este método se implementó mediante una encuesta estructurada la cual permitió medir la percepción de los usuarios respecto a la utilidad y eficiencia de la aplicación web que se desarrollara.

La unión de ambos enfoques permitió obtener una visión integral de la situación actual del negocio.

4.2 TIPO DE INVESTIGACIÓN

4.2.1 Investigación Bibliográfica

Este tipo de investigación se utilizó para recopilar información teórica relacionada con las aplicaciones web progresivas (PWA), metodologías ágiles y gestión de pedidos. Para ello se revisaron libros, artículos científicos, tesis y documentos académicos que permitieron fundamentar el desarrollo del proyecto y sustentar el marco teórico de la investigación.

4.2.2 Investigación de Campo

Como parte del proceso investigativo, se desarrolló una investigación de campo en la cual se realizaron entrevistas semiestructuradas y encuestas estructuradas al propietario y al personal del Café Restaurante “Productos Carlos Gerardo”, entre los cuales se incluyen meseros, cocineros y cajeros. Durante esta investigación se recopilaron opiniones, experiencias y sugerencias relacionadas con la gestión actual de pedidos, cobros y la comunicación interna.

Las entrevistas se utilizaron para obtener información cualitativa, la cual está relacionada con los procesos manuales existentes y las necesidades del negocio, lo que facilitó la identificación de los requerimientos funcionales del sistema. Por otra parte, las encuestas estructuradas facilitaron el análisis de la percepción de los usuarios con respecto a la utilidad, facilidad y eficiencia de la aplicación web progresiva propuesta.

4.2.3 Investigación tecnológica

Este tipo de investigación se orientó al análisis, selección y aplicación de herramientas tecnológicas necesarias para el desarrollo de la aplicación web progresiva (PWA) destinada a la gestión de pedidos del Café Restaurante "Productos Carlos Gerardo". A través de esta investigación se evaluaron diferentes tecnologías, lenguajes de programación, frameworks y gestores de bases de datos que permitieran construir una solución eficiente, segura y adaptable a las necesidades del negocio.

Asimismo, se analizaron las características y funcionalidades que ofrecen las aplicaciones web progresivas, tales como la instalación en dispositivos, las notificaciones push y la capacidad de ejecutarse en distintos dispositivos mediante un navegador web. Esto permitió seleccionar las tecnologías más adecuadas para el diseño e implementación del sistema, garantizando su correcto funcionamiento, accesibilidad y facilidad de uso para los usuarios del restaurante.

4.3 TÉCNICAS DE INVESTIGACIÓN

4.3.1 Revisión Documental

La revisión documental es una técnica de investigación que consiste en el análisis y estudio de documentos y fuentes de información previamente publicadas, con el propósito de obtener datos relevantes que contribuyan al desarrollo de la investigación. Entre estas fuentes se incluyen libros, artículos científicos, tesis, revistas académicas y documentos especializados relacionados con el tema de estudio.

En el presente proyecto, esta técnica se utilizó para recopilar información teórica acerca de las aplicaciones web progresivas (PWA), metodologías ágiles y gestión de pedidos en establecimientos gastronómicos. La revisión de estos documentos permitió sustentar el marco teórico de la investigación, comprender conceptos fundamentales y apoyar el desarrollo de la propuesta tecnológica planteada.

4.3.2 Entrevista

La entrevista semiestructurada es una técnica de investigación cualitativa, la cual consta de una guía de preguntas previamente definidas, manteniendo la flexibilidad necesaria para profundizar en las respuestas proporcionadas por las personas entrevistadas.

Esta entrevista se realizó con el objetivo de comprender los procesos actuales de gestión de pedidos, cobros y comunicación interna, ya que facilita la recopilación de opiniones, experiencias y sugerencias, permitiendo así analizar de manera detallada el contexto real del problema.

4.3.3 Encuesta

La encuesta estructurada es una técnica de investigación cuantitativa en la cual se utilizan preguntas cerradas, como opciones múltiples o escalas, para obtener información precisa y fácil de analizar. Por lo que se diseñó una encuesta de forma estandarizada que garantice que todos los encuestados respondan bajo las mismas condiciones, lo que facilitó la comparación y análisis cuantitativo de los resultados.

Esta técnica de investigación resultó adecuada para medir la percepción de los usuarios respecto a la utilidad, facilidad de uso y eficiencia de la aplicación.

4.4 TÉCNICAS ESPECIFICAS

4.4.1 Pruebas funcionales

Las pruebas funcionales constituyen una técnica utilizada para verificar que cada una de las funcionalidades del sistema opere de acuerdo con los requerimientos establecidos. Estas pruebas se enfocan en evaluar el comportamiento del sistema, comprobando que las entradas, procesos y salidas funcionen correctamente según lo esperado.

En el presente proyecto, las pruebas funcionales se realizaron mediante la elaboración y ejecución de casos de prueba desarrollados en cada iteración del proceso de desarrollo. Los cuales permitieron validar las funcionalidades principales del sistema, tales como el registro y gestión de pedidos, procesamiento de pagos, comunicación entre cocina y meseros, así como la visualización de información dentro de la aplicación.

La aplicación de estas pruebas permitió identificar posibles errores, verificar el correcto funcionamiento del sistema y asegurar que los requerimientos funcionales se cumplan de

manera adecuada, contribuyendo así a mejorar la calidad y confiabilidad de la aplicación desarrollada.

4.4.2 Metodología de desarrollo del sistema

4.4.2.1 Aplicación de la metodología XP

Para el desarrollo de la aplicación se seleccionó la metodología eXtreme Programming (XP) debido a que permitió trabajar de forma iterativa y flexible, considerando que el sistema requiere cambios frecuentes, verificaciones constantes por parte de los usuarios y la entrega de productos funcionales que puedan ser aprobados en un entorno real de trabajo.

La metodología se aplicó mediante iteraciones cortas las cuales se realizaron siguiendo las fases de planificación, diseño, codificación y pruebas, lo que permitió la entrega de productos funcionales al cliente después de finalizar cada iteración para así realizar ajustes o mejoras continuas conforme a sus necesidades.

Esto permitió la construcción incremental de la aplicación asegurando que cada módulo desarrollado cumpliera los requerimientos del negocio.

Además, debido a que esta metodología es utilizada para equipos de desarrollo medianos y grandes, se realizó una adaptación, ya que el desarrollo del proyecto fue realizado por un equipo reducido de dos personas. En donde el estudiante desarrollador asumió las actividades principales relacionadas con la planificación, diseño, codificación y pruebas del sistema. Por otro lado, el docente tutor participó como guía durante el proceso de desarrollo, en donde brindó orientación técnica y metodológica durante las diferentes fases del proyecto; además, realizó el seguimiento del progreso del proyecto y el cumplimiento de las iteraciones.

Esta adaptación permitió mantener los principios de la metodología XP, fomentando la retroalimentación constante, el control del progreso del proyecto y la mejora continua del sistema desarrollado.

4.4.2.1.1 Justificación de la Metodología XP

La metodología Extreme Programming (XP) fue seleccionada para el desarrollo del sistema debido a que permite trabajar mediante ciclos de desarrollo cortos e iterativos, facilitando la adaptación a cambios durante el proceso de construcción del software. Este enfoque resulta adecuado para el presente proyecto, ya que el sistema de gestión de pedidos requiere ajustes continuos basados en las necesidades reales del Café Restaurante “Productos Carlos Gerardo”.

Además, XP promueve la comunicación constante entre desarrolladores y usuarios, lo cual permite validar cada funcionalidad desarrollada y realizar mejoras progresivas en el sistema. Esta característica es especialmente importante en el desarrollo de la aplicación web progresiva (PWA), ya que permite probar los módulos implementados en un entorno real de trabajo y asegurar que cumplan con los requerimientos del negocio.

Con base en las características de una aplicación web progresiva [11], el sistema creado cumple con varias de ellas. En primer lugar, emula la opción de instalación en un dispositivo móvil, puesto que, puede ser añadida a la pantalla principal y ejecutarse de forma similar a una app nativa. También tiene compatibilidad en múltiples plataformas, ya que, se puede acceder desde diferentes dispositivos a través de un navegador web sin necesidad de un sistema operativo específico. Además, facilita su distribución mediante un enlace web, lo que permite a los usuarios acceder al sistema sin requerir descarga desde tiendas de aplicaciones.

Finalmente, la aplicación incorpora notificaciones push que permiten mantener a los usuarios informados sobre eventos relevantes dentro del sistema, promoviendo una comunicación más ágil y eficaz. Sin embargo, la funcionalidad de acceso sin conexión no está disponible en esta versión del sistema, ya que, su operación requiere estar conectado al servidor y a la base de datos para gestionar los pedidos y mantener la información actualizada en tiempo real.

4.4.2.1.2 Fases de la Metodología XP

La metodología XP está estructurada en varias fases que se ejecutan de manera iterativa, en cada fase se generan artefactos que permiten organizar, desarrollar y validar el sistema.

A continuación, se describen las fases de la metodología XP y los artefactos utilizados en cada una de ellas.

4.4.2.1.3 Planificación

Durante esta fase se realizaron reuniones con el propietario y los trabajadores del restaurante para identificar y organizar las necesidades reales del establecimiento. En base a la información obtenida se definieron las historias de usuario, se priorizaron funcionalidades y se elaboró el Release Planning el cual permitió estructurar las entregas del sistema de manera progresiva.

4.4.2.1.3.1 Historias de usuario

Se utilizaron historias de usuario para describir las funcionalidades del sistema, las cuales se realizaron en las reuniones con los usuarios, ya que son ellos quienes deciden qué tareas realizará la aplicación. Estas historias son utilizadas como una herramienta para dar a conocer

de forma clara los requerimientos del sistema que debe cumplir el desarrollador del equipo en cada iteración.

A continuación, se mostrará el modelo de historia de usuario que se usará para describir cada apartado del sistema.

Tabla 4.1. Historia de usuario detallada.

Historia de Usuario			
Número	Identificador único.	**Usuario**	Tipo de usuario o rol que solicita la funcionalidad.
Nombre	Título corto y descriptivo de la historia de usuario.		
Prioridad	Nivel de importancia.	**Puntos Estimados**	Estimación del esfuerzo necesario para implementar la historia de usuario.
Responsable	Persona o encargado de desarrollar la historia.	**Iteración Asignada**	Ciclo de desarrollo en el que se implementará la historia.
Descripción	Como (usuario), quiero (acción), para (beneficio).		

4.4.2.1.3.2 Diagrama caso de uso

Los diagramas de caso de uso se emplearon para identificar las funcionalidades del sistema desde la perspectiva de los usuarios, estos diagramas permitieron definir qué acciones puede realizar cada tipo de usuario y como interactúan con el sistema para cumplir sus tareas diarias dentro del restaurante. De esta manera, los diagramas permitieron representar la estructura funcional del sistema estableciendo las responsabilidades de cada rol como administrador, mesero, cocinero, cajero y clientes.

Aunque los diagramas de caso de uso se utilizan principalmente en metodologías tradicionales de desarrollo de software en el presente proyecto fueron empleados como una herramienta de apoyo durante la fase de planificación. Esto permitió complementar las historias de usuario definidas en la metodología XP, facilitando la visualización de los requerimientos funcionales

y asegurando que todas las funcionalidades del sistema estuvieran correctamente representadas antes de su implementación.

4.4.2.1.4 Diseño

En la fase de diseño se define la estructura visual, funcional y lógica del sistema para asegurar que la aplicación sea fácil de usar, intuitiva y adecuada para los procesos diarios del Café Restaurante "Productos Carlos Gerardo". Durante esta fase se priorizó la simplicidad debido a que los usuarios finales no cuentan con experiencia previa en el uso de sistemas de gestión de pedidos.

El diseño se realizó de forma incremental, permitiendo realizar ajustes conforme se validaban las propuestas con los trabajadores del restaurante, y así cumplir con los requisitos del sistema obtenidas en las historias de usuario.

Para esta fase se desarrollaron prototipos de interfaces, diagramas de caso de uso y diseño de base de datos.

4.4.2.1.4.1 Prototipo de interfaces

Se desarrollaron prototipos de las interfaces del sistema utilizando la herramienta Figma, con el fin de visualizar la distribución de los elementos, la navegación y las acciones disponibles para cada tipo de usuario. Estos prototipos permitieron representar la estructura visual del sistema, facilitando así la organización de la información en pantalla. Además, estos prototipos fueron realizados de manera iterativa para luego ser validados por los trabajadores del restaurante, permitiendo realizar ajustes antes de la fase de codificación y reduciendo posibles errores de usabilidad.

4.4.2.1.4.2 Diseño de base de datos

El diseño de la base de datos tuvo como objetivo definir la estructura de almacenamiento de la información que maneja el sistema para así garantizar la organización de los datos generados durante el proceso de gestión de pedidos.

En este apartado permitió representar la estructura lógica del sistema identificando las entidades principales y las relaciones entre ellas, evitando la duplicidad de información y asegurando un acceso eficiente a los datos registrados en el sistema. En el proyecto se diseñaron tablas para usuarios, mesa, producto, pedido, detalle del pedido, pago, comprobante, notificación y mensaje.

4.4.2.1.5 Codificación

Esta fase se realizó siguiendo la metodología XP priorizando la entrega continua de funcionalidades de forma incremental, permitiendo que el sistema sea funcional desde las primeras iteraciones.

4.4.2.1.5.1 Código fuente

El código fuente representa el desarrollo del sistema utilizando tecnologías web adecuadas para la creación de una aplicación, siguiendo las historias de usuario para cumplir con las funcionalidades específicas de cada una.

4.4.2.1.6 Prueba

La fase de pruebas permitió verificar el correcto funcionamiento del sistema y comprobar que cada historia de usuario cumpla con los requerimientos establecidos. Las pruebas se realizan de forma continua, acompañando cada fase de codificación; esta fase es fundamental para asegurar que el sistema sea confiable, fácil de usar y adecuado para los usuarios.

4.4.2.1.6.1 Casos de prueba

Los casos de prueba se definieron para comprobar el funcionamiento del sistema mediante cada historia de usuario, describiendo las acciones que realizará dicha historia y los resultados esperados para luego proceder a ejecutar las mencionadas acciones y evidenciar el resultado mediante una captura.

Tabla 4.2. Plantilla de caso de prueba.

Campo	Descripción
Código	Código del Caso de Prueba
Historia de Usuario	Identificar único de la historia de usuario
Nombre del Caso de Prueba	Nombre descriptivo
Objetivo	Validar la funcionalidad descrita en la historia de usuario
Usuario	Rol que ejecuta la prueba
Precondiciones	Condiciones necesarias antes de ejecutar la prueba
Datos de prueba	Información que se ingresa en el sistema
Pasos a ejecutar	Secuencia de acciones a realizar

Campo	Descripción
Resultado esperado	Comportamiento esperado del sistema
Resultado obtenido	Resultado real al ejecutar la prueba
Evidencia	Captura de pantalla

4.5 HERRAMIENTAS DE DESARROLLO

Para el desarrollo de esta aplicación se requiere del uso adecuado de herramientas tecnológicas que permitan construir el sistema de manera eficiente y segura; además, estas herramientas constituyen el soporte técnico, ya que facilitan la implementación de funcionalidades, la gestión de la información y la interacción entre los distintos componentes de la aplicación.

La correcta selección de estas herramientas se realiza considerando criterios como compatibilidad tecnológica, rendimiento, facilidad de mantenimiento, seguridad y adecuación a los requerimientos del sistema, ya que no solo permiten la creación del sistema, sino que también influyen directamente en su calidad final.

4.5.1 Herramientas de desarrollo

Tabla 4.3. Herramienta de desarrollo del sistema.

Categoría	Herramienta	Versión	Descripción
Lenguaje de programación	Python	3.12.x	Lenguaje principal utilizado para el desarrollo del backend del sistema.
Framework backend	Django	5.0.x	Framework web que permite el desarrollo rápido, seguro y estructurado del sistema.
Comunicación en tiempo real	Django Channels	4.1.x	Extensión de Django utilizada para implementar comunicación en tiempo real mediante WebSockets.
Servidor ASGI	Daphne	4.1.x	Servidor ASGI utilizado para manejar solicitudes HTTP y WebSockets.
Base de datos	PostgreSQL	17.x	Sistema gestor de base de datos relacional utilizado para el almacenamiento de la información.

Categoría	Herramienta	Versión	Descripción
Frontend	HTML5	—	Lenguaje de marcado utilizado para estructurar las interfaces del sistema.
Estilos	CSS3	—	Lenguaje utilizado para el diseño visual y presentación de las interfaces.
Interactividad	JavaScript	ES6	Lenguaje utilizado para la interacción dinámica del sistema en el navegador.
Framework CSS	Bootstrap	5.3.x	Framework utilizado para el diseño responsivo y componentes visuales.
Librería JS	jQuery	3.7.x	Librería utilizada para facilitar la manipulación del DOM y eventos.
Generación de PDFs	ReportLab	4.x	Librería utilizada para la generación de comprobantes en formato PDF.
Control de versiones	Git	2.4x.x	Herramienta utilizada para el control de versiones del código fuente.
Entorno de desarrollo	Visual Studio Code	1.9x.x	Editor de código utilizado para el desarrollo del sistema.
Repositorio remoto	GitHub	—	Plataforma utilizada para el almacenamiento y gestión del repositorio del proyecto.

4.6 POBLACIÓN Y MUESTRA

4.6.1 Población

Para esta investigación, la población considerada está conformada por los trabajadores del Café Restaurante “Productos Carlos Gerardo” incluyendo meseros, cocineros, cajeros y el propietario/administrador del establecimiento los cuales son 12 en total. Estos beneficiarios participan directamente en los procesos operativos del restaurante y son considerados los usuarios finales del sistema.

4.6.2 Muestra

Debido al tamaño reducido de la población, se empleó un muestreo censal considerando al total de los trabajadores como muestra de estudio lo que permitió obtener información directa y representativa para la realización del sistema, lo cual a su vez facilitó la validación del sistema en un entorno real, garantizando así resultados confiables y acordes a las necesidades del negocio.

5. ANÁLISIS Y DISCUSIÓN DE RESULTADOS

5.1 RESULTADOS DE LA ENTREVISTA

La entrevista semiestructurada que se aplicó al propietario y trabajadores del Café Restaurante "Productos Carlos Gerardo" se puede visualizar en el Anexo A la cual se realizó con el propósito de conocer cómo se manejan actualmente los procesos operativos en las diferentes áreas del restaurante.

A partir de las respuestas obtenidas, se identificó que la gestión de pedidos se realiza de forma manual, principalmente mediante el uso de libretas y comunicación verbal, ya que los meseros anotan los pedidos en papel y los llevan a la cocina, mientras que el cobro se efectúa de manera independiente en el área de caja. Por lo que no existe un sistema que integre toda la información, lo que ocasiona que cada área trabaje de forma separada y sin un control.

Los entrevistados señalaron que este método de trabajo genera varios problemas, como la confusión en los pedidos, errores en la información registrada y demoras en la atención; lo ocasiona reclamos por parte de los clientes. También mencionaron que esto ocasiona estrés en el ambiente de trabajo, afectando así el desempeño del personal.

En cuanto a la posibilidad de automatizar procesos mediante un sistema, los entrevistados consideran que permitirá mejorar significativamente la gestión operativa del restaurante, como el control de pedidos, comunicación de los pedidos a realizar en cocina y mejorando el registro de cobros. En general, todos coincidieron en que un sistema de comunicación en tiempo real ayudaría a mejorar la eficiencia del trabajo diario, minimizar errores y optimizar la coordinación entre las diferentes áreas.

En base a los resultados obtenidos en la entrevista, se concluye que la implementación de una aplicación web progresiva para la gestión de pedidos en tiempo real responde a una necesidad

real del Café Restaurante “Productos Carlos Gerardo”, ya que permitirá mejorar la gestión operativa, ofreciendo así un mejor servicio a los clientes.

5.2 RESULTADOS DE LA ENCUESTA

El formulario de las preguntas realizadas al personal del restaurante se visualiza en el Anexo B.

5.2.1 Pregunta 1: Cargo que desempeña en el restaurante

Tabla 5.1. Cargo que desempeña en el restaurante

Cargo	Frecuencia (fi)	Frecuencia relativa (fr)	Porcentaje (%)
Mesero	4	0,333	33,3%
Cocinero	4	0,333	33,3%
Cajero	3	0,25	25%
Propietario	1	0,083	8,3%
Total	**12**	**1,00**	**100%**

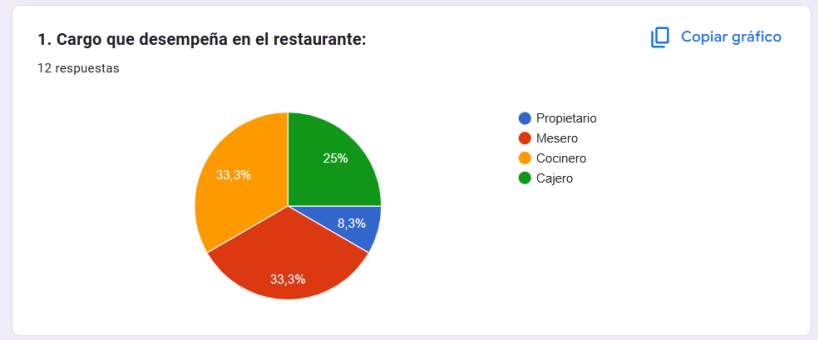


Figura 5.1. Cargo que desempeña en el restaurante

Análisis:

- El 33,3% de los encuestados (4 personas) desempeña el cargo de mesero, mientras que otro 33,3% (4 personas) corresponde al cargo de cocinero, siendo estos los roles con mayor representación dentro del restaurante.
- El 25% de los participantes (3 personas) ocupa el cargo de cajero.
- El 8,3% de los encuestados (1 persona) corresponde al propietario del establecimiento.

Los resultados indican que la mayoría de los encuestados pertenece al personal operativo (meseros y cocineros), quienes están directamente involucrados en la toma y preparación de pedidos.

5.2.2 Pregunta 2: Tiempo que lleva trabajando en el restaurante

Tabla 5.2. Tiempo que lleva trabajando en el restaurante

Tiempo de servicio	Frecuencia (fi)	Frecuencia relativa (fr)	Porcentaje (%)
Menos de 6 meses	1	0,083	8,3%
6 meses – 1 año	3	0,25	25%
Más de 1 año	8	0,667	66,7%
Total	**12**	**1,00**	**100%**

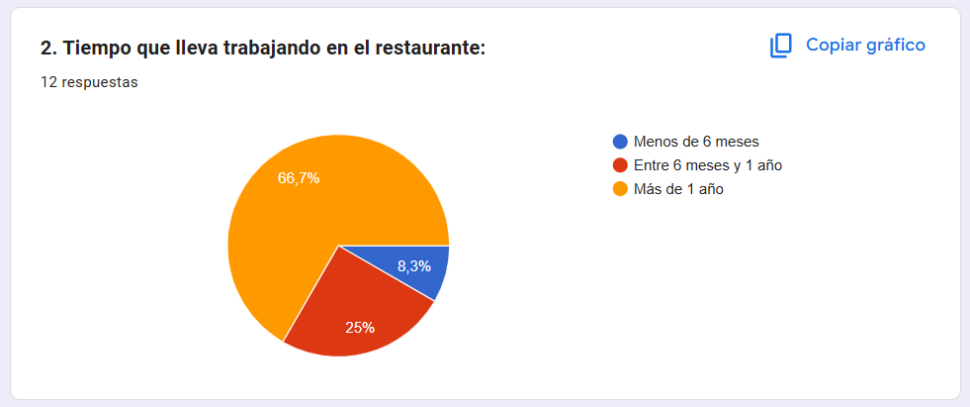


Figura 5.2. Tiempo que lleva trabajando en el restaurante

Análisis:

- El 66,7% de los encuestados (8 personas) indicó que lleva más de un año trabajando en el restaurante, lo que refleja una alta estabilidad laboral y experiencia en los procesos internos.
- El 25% (3 personas) señaló que ha trabajado entre 6 meses y 1 año, mientras que solo el 8,3% (1 persona) tiene menos de 6 meses en el establecimiento.

Los resultados indican que la mayoría del personal cuenta con amplia experiencia, lo cual es favorable para la implementación de un sistema web.

5.2.3 Pregunta 3: ¿Considera necesario implementar un sistema web para la gestión de pedidos en el restaurante?

Tabla 5.3. ¿Considera necesario implementar un sistema web para la gestión de pedidos en el restaurante?

Respuesta	Frecuencia (fi)	Frecuencia relativa (fr)	Porcentaje (%)
Sí	12	1,00	100%
No	0	0,00	0%
Total	**12**	**1,00**	**100%**

Figura 5.3. ¿Considera necesario implementar un sistema web para la gestión de pedidos en el restaurante?

Análisis:

- El 100% de los encuestados (12 personas) respondió sí, indicando que consideran necesaria la implementación de un sistema web para la gestión de pedidos.

Los resultados demuestran una aceptación total por parte del personal, lo que respalda la viabilidad del proyecto y confirma que el sistema responde a una necesidad real dentro del restaurante.

5.2.4 Pregunta 4: ¿Cree que un sistema web reduciría los errores en la toma de pedidos?

Tabla 5.4. ¿Cree que un sistema web reduciría los errores en la toma de pedidos?

Respuesta	Frecuencia (fi)	Frecuencia relativa (fr)	Porcentaje (%)
Sí	9	0,75	75%
No	3	0,25	25%
Total	**12**	**1,00**	**100%**

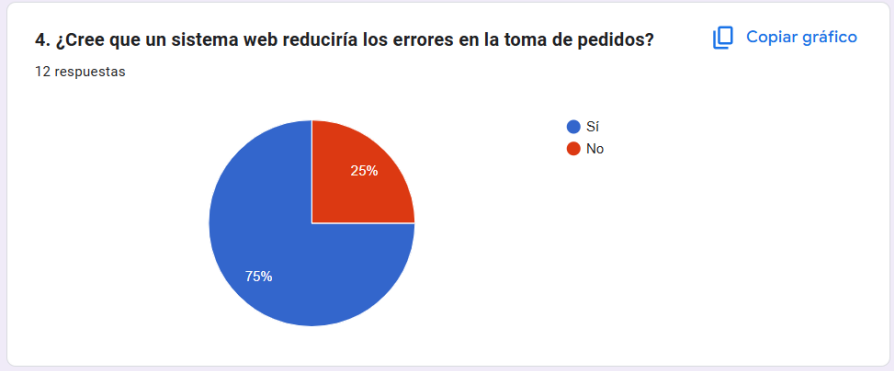


Figura 5.4. ¿Cree que un sistema web reduciría los errores en la toma de pedidos?

Análisis:

- El 75% de los encuestados (9 personas) considera que un sistema web sí reduciría los errores, mientras que el 25% (3 personas) opina lo contrario.

Los resultados indican que la mayoría de los encuestados consideran que el sistema web permitirá disminuir errores en la toma de pedidos.

5.2.5 Pregunta 5: ¿Un sistema con comunicación en tiempo real mejoraría la coordinación entre las áreas?

Tabla 5.5. ¿Un sistema con comunicación en tiempo real mejoraría la coordinación entre las áreas?

Respuesta	Frecuencia (fi)	Frecuencia relativa (fr)	Porcentaje (%)
Sí	10	0,833	83,3%
No	2	0,167	16,7%
Total	**12**	**1,00**	**100%**

Figura 5.5. ¿Un sistema con comunicación en tiempo real mejoraría la coordinación entre las áreas?

Análisis:

- El 83,3% de los encuestados (10 personas) respondió sí, mientras que el 16,7% (2 personas) considera que no tendría un impacto significativo.

Los resultados indican que la mayoría del personal reconoce la importancia de la comunicación en tiempo real para optimizar la coordinación entre cocina, meseros y caja, reduciendo tiempos de espera y confusiones.

5.2.6 Pregunta 6: ¿Considera importante que el sistema sea fácil de usar?

Tabla 5.6. ¿Considera importante que el sistema sea fácil de usar?

Respuesta	Frecuencia (fi)	Frecuencia relativa (fr)	Porcentaje (%)
Sí	11	0,917	91,7%
No	1	0,083	8,3%
Total	**12**	**1,00**	**100%**

Figura 5.6. ¿Considera importante que el sistema sea fácil de usar?

Análisis:

- El 91,7% de los encuestados (11 personas) considera que sí es importante, mientras que solo el 8,3% (1 persona) respondió no.

Los resultados resaltan la necesidad de diseñar un sistema intuitivo y amigable, que facilite su adopción por parte de todo el personal.

5.2.7 Pregunta 7: ¿Cree que aprender a usar el sistema sería sencillo para el personal?

Tabla 5.7. ¿Cree que aprender a usar el sistema sería sencillo para el personal?

Respuesta	Frecuencia (fi)	Frecuencia relativa (fr)	Porcentaje (%)
Sí	8	0,667	66,7%
No	4	0,333	33,3%
Total	**12**	**1,00**	**100%**

Figura 5.7. ¿Cree que aprender a usar el sistema sería sencillo para el personal?

Análisis:

- El 66,7% de los encuestados (8 personas) considera que sí sería sencillo, mientras que el 33,3% (4 personas) opina que podría resultar complicado.

Los resultados indican que, aunque existe una percepción mayoritariamente positiva, será importante incluir capacitación básica para garantizar un uso adecuado del sistema por todo el personal.

5.2.8 Pregunta 8: ¿El uso del sistema permitiría atender a los clientes de forma más rápida?

Tabla 5.8. ¿El uso del sistema permitiría atender a los clientes de forma más rápida?.

Respuesta	Frecuencia (fi)	Frecuencia relativa (fr)	Porcentaje (%)
Sí	9	0,750	75%
No	3	0,250	25%
Total	**12**	**1,00**	**100%**

Figura 5.8. ¿El uso del sistema permitiría atender a los clientes de forma más rápida?.

Análisis:

- El 75% de los encuestados (9 personas) considera que sí, mientras que el 25% (3 personas) respondió no.

Los resultados indican que la mayoría de los encuestados reconoce que el sistema web permitirá agilizar los procesos de atención, reduciendo tiempos de espera y mejorando la experiencia del cliente.

5.2.9 Pregunta 9: ¿Considera que el sistema mejoraría la organización del trabajo diario?.

Tabla 5.9. ¿Considera que el sistema mejoraría la organización del trabajo diario?.

Respuesta	Frecuencia (fi)	Frecuencia relativa (fr)	Porcentaje (%)
Sí	11	0,917	91,7%
No	1	0,083	8,3%
Total	**12**	**1,00**	**100%**

Figura 5.9. ¿Considera que el sistema mejoraría la organización del trabajo diario?.

Análisis:

- El 91,7% de los encuestados (11 personas) respondió sí, mientras que solo el 8,3% (1 persona) considera que no.

Este resultado evidencia que el sistema web contribuirá significativamente a una mejor organización de las actividades diarias.

5.2.10 Pregunta 10: ¿Cree que el sistema sería eficiente para las actividades que realiza?

Tabla 5.10. ¿Cree que el sistema sería eficiente para las actividades que realiza?

Respuesta	Frecuencia (fi)	Frecuencia relativa (fr)	Porcentaje (%)
Sí	9	0,750	75%
No	3	0,250	25%
Total	**12**	**1,00**	**100%**

Figura 5.10. ¿Cree que el sistema sería eficiente para las actividades que realiza?

Análisis:

- El 75% de los encuestados (9 personas) considera que el sistema sería eficiente, mientras que el 25% (3 personas) opina que no.

Los resultados indican que el sistema web apoyará el desempeño de las actividades laborales, optimizando procesos y reduciendo la carga operativa

5.2.11 Pregunta 11: ¿El sistema reduciría el uso de procesos manuales?

Tabla 5.11. ¿El sistema reduciría el uso de procesos manuales?

Respuesta	Frecuencia (fi)	Frecuencia relativa (fr)	Porcentaje (%)
Sí	10	0,833	83,3%
No	2	0,167	16,7%
Total	**12**	**1,00**	**100%**

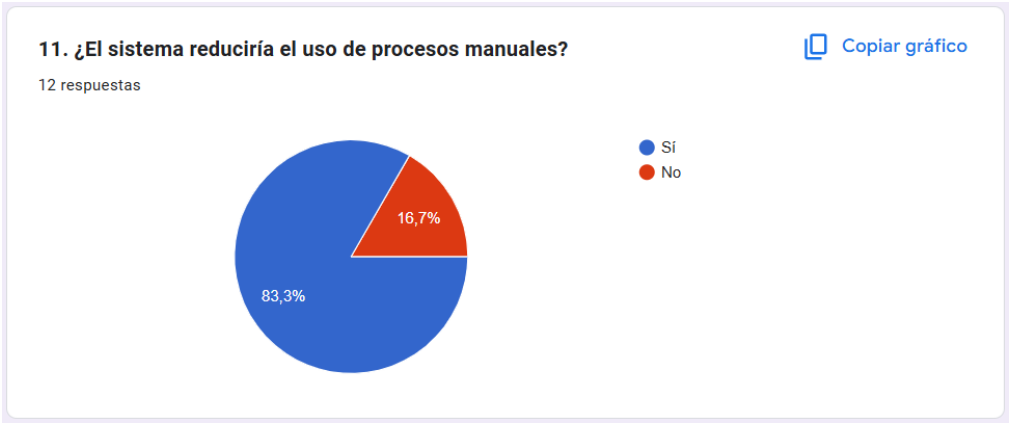


Figura 5.11. ¿El sistema reduciría el uso de procesos manuales?

Análisis:

- El 83,3% de los encuestados (10 personas) considera que sí reduciría los procesos manuales, mientras que el 16,7% (2 personas) respondió no.

Los resultados indican que la implementación del sistema web permitirá automatizar tareas, disminuir el uso de registros manuales y mejorar la eficiencia operativa.

5.2.12 Pregunta 12: ¿Estaría dispuesto/a a utilizar el sistema en sus actividades diarias?

Tabla 5.12. ¿Estaría dispuesto/a a utilizar el sistema en sus actividades diarias?

Respuesta	Frecuencia (fi)	Frecuencia relativa (fr)	Porcentaje (%)
Sí	10	0,833	83,3%
No	2	0,167	16,7%
Total	**12**	**1,00**	**100%**

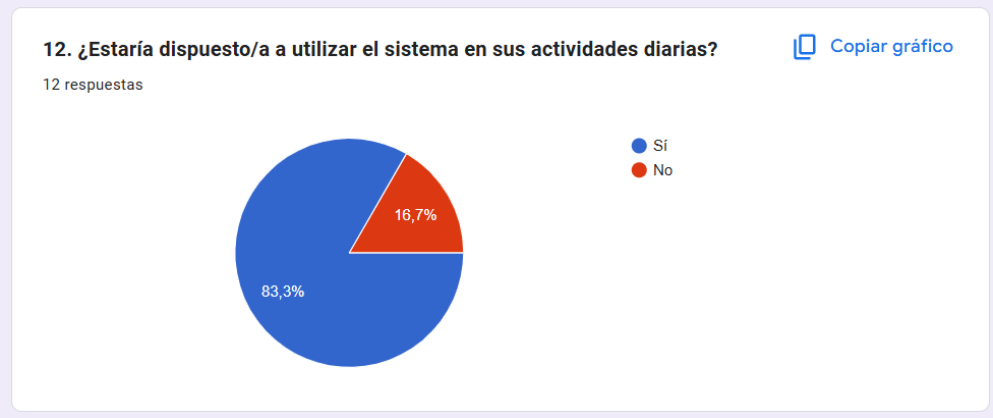


Figura 5.12. ¿Estaría dispuesto/a a utilizar el sistema en sus actividades diarias?

Análisis:

- El 83,3% de los encuestados (10 personas) manifestó que sí estaría dispuesto/a a utilizar el sistema en sus actividades diarias, mientras que el 16,7% (2 personas) respondió no.

Los resultados evidencian una alta predisposición del personal hacia el uso del sistema web propuesto, lo que representa un factor clave para su implementación exitosa.

5.3 RESULTADO DE LA APLICACIÓN DE LA METODOLOGÍA XP

5.3.1 Roles

Para el avance del sistema se definieron los siguientes roles:

Tabla 5.13. Plantilla de roles de la metodología XP.

Rol	Responsable	Funciones
Programador	Desarrollador	Análisis, diseño, codificación y pruebas del sistema
Cliente	Propietario del restaurante	Definición de requerimientos y validación de funcionalidades
Tester	Personal del restaurante	Pruebas funcionales
Tracker	Investigador	Seguimiento del avance de historias de usuario
Coach	Docente tutor	Orientación metodológica y revisión técnica

Utilizando la plantilla de roles de la metodología XP Tabla 5.13. se asignaron funciones y responsables para cada rol, lo cual se muestra a continuación.

Tabla 5.14. Roles de la metodología XP.

Rol	Responsable	Funciones
Programador	Katherin Pico	Levantamiento de requerimientos . Análisis de las historias de usuario. Desarrollo del sistema.
Cliente	Sr. Carlos Morales	Proveer información sobre los requerimientos del sistema. Verificar funcionalidades implementadas.
Tester	Katherin Pico	Documentar casos de prueba realizados. Verificar el correcto funcionamiento del sistema.

Rol	Responsable	Funciones
Tracker	MSc. . Victor Medina	Dar seguimiento al avance del equipo de desarrollo y proporcionar retroalimentación continua.
Coach	MSc. . Victor Medina	Guiar al equipo de desarrollo en la correcta implementación de la metodología XP

5.3.2 Historias de usuario

Para realizar las historias de usuario se utilizó la plantilla descrita en la tabla 4.1, la cual permitió estructurar de manera clara los requerimientos del sistema desde la perspectiva de los usuarios.

Tabla 5.15. HU: Crear usuarios.

Historia de usuario			
Número	HU01	**Usuario**	Administrador
Nombre	Crear usuarios		
Prioridad	Alta	**Puntos Estimados**	3
Responsable	Katherin Pico	**Iteración Asignada**	Iteración 1
Descripción	Como administrador, quiero crear usuarios del sistema, para que el personal del restaurante pueda acceder a la aplicación según su función.		

Tabla 5.16. HU: Actualizar usuarios.

Historia de usuario			
Número	HU02	**Usuario**	Administrador
Nombre	Actualizar usuarios		
Prioridad	Alta	**Puntos Estimados**	2
Responsable	Katherin Pico	**Iteración Asignada**	Iteración 1

Descripción	Como administrador, quiero actualizar la información de los usuarios, para mantener los datos del personal correctamente registrados.

Tabla 5.17. HU: Visualizar usuarios.

Historia de usuario			
Número	HU03	**Usuario**	Administrador
Nombre	Visualizar usuarios		
Prioridad	Alta	**Puntos Estimados**	2
Responsable	Katherin Pico	**Iteración Asignada**	Iteración 1
Descripción	Como administrador, quiero visualizar el listado de usuarios, para administrar al personal del restaurante de forma ordenada.		

Tabla 5.18. HU: Desactivar usuarios.

Historia de usuario			
Número	HU04	**Usuario**	Administrador
Nombre	Desactivar usuarios		
Prioridad	Alta	**Puntos Estimados**	2
Responsable	Katherin Pico	**Iteración Asignada**	Iteración 1
Descripción	Como administrador, quiero desactivar usuarios del sistema, para evitar que personal inactivo siga accediendo a la aplicación.		

Tabla 5.19. Registrar mesas.

Historia de usuario			
Número	HU05	**Usuario**	Administrador
Nombre	Registrar mesas		
Prioridad	Alta	**Puntos Estimados**	3

Responsable	Katherin Pico	**Iteración Asignada**	Iteración 2
Descripción	Como administrador, quiero registrar las mesas del restaurante, para que los meseros puedan asignarlas correctamente a los clientes.		

Tabla 5.20. Registrar productos del menú.

Historia de usuario			
Número	HU06	**Usuario**	Administrador
Nombre	Registrar productos del menú		
Prioridad	Alta	**Puntos Estimados**	3
Responsable	Katherin Pico	**Iteración Asignada**	Iteración 3
Descripción	Como administrador, quiero registrar los productos del menú con su precio, para que puedan ser ofrecidos correctamente a los clientes.		

Tabla 5.21. Visualizar reportes de ventas.

Historia de usuario			
Número	HU07	**Usuario**	Administrador
Nombre	Visualizar reportes de ventas		
Prioridad	Media	**Puntos Estimados**	3
Responsable	Katherin Pico	**Iteración Asignada**	Iteración 8
Descripción	Como administrador, quiero visualizar reportes de ventas, para controlar los ingresos del restaurante.		

Tabla 5.22. Editar productos del menú.

Historia de usuario			
Número	HU08	**Usuario**	Administrador

Nombre	Editar productos del menú		
Prioridad	Alta	**Puntos Estimados**	3
Responsable	Katherin Pico	**Iteración Asignada**	Iteración 3
Descripción	Como administrador, quiero editar la información de los productos del menú, para actualizar precios, nombres o descripciones cuando sea necesario.		

Tabla 5.23. Desactivar productos del menú.

Historias de usuario			
Número	HU09	**Usuario**	Administrador
Nombre	Desactivar productos del menú		
Prioridad	Media	**Puntos Estimados**	2
Responsable	Katherin Pico	**Iteración Asignada**	Iteración 3
Descripción	Como administrador, quiero desactivar productos del menú, para que no se muestren cuando ya no se encuentren disponibles de forma permanente.		

Tabla 5.24. Visualizar reportes por tipo de pedido.

Historias de usuario			
Número	HU10	**Usuario**	Administrador
Nombre	Visualizar reportes por tipo de pedido		
Prioridad	Media	**Puntos Estimados**	3
Responsable	Katherin Pico	**Iteración Asignada**	Iteración 8
Descripción	Como administrador, quiero visualizar reportes separados de pedidos de domicilio y restaurante, para analizar el desempeño de cada tipo de servicio.		

Tabla 5.25. Visualizar reporte unificado de cobros.

Historias de usuario			
Número	HU11	**Usuario**	Administrador
Nombre	Visualizar reporte unificado de cobros		
Prioridad	Media	**Puntos Estimados**	3
Responsable	Katherin Pico	**Iteración Asignada**	Iteración 8
Descripción	Como administrador, quiero visualizar un reporte unificado de todos los cobros del restaurante, para tener un control general de los ingresos.		

Tabla 5.26. Editar perfil administrador.

Historias de usuario			
Número	HU12	**Usuario**	Administrador
Nombre	Editar perfil administrador		
Prioridad	Media	**Puntos Estimados**	2
Responsable	Katherin Pico	**Iteración Asignada**	Iteración 9
Descripción	Como administrador, quiero editar mi perfil, para mantener actualizada mi información personal dentro del sistema.		

Tabla 5.27. Editar perfil mesero.

Historia de usuario			
Número	HU13	**Usuario**	Mesero
Nombre	Editar perfil mesero		
Prioridad	Media	**Puntos Estimados**	2
Responsable	Katherin Pico	**Iteración Asignada**	Iteración 9
Descripción	Como mesero, quiero editar mi perfil, para mantener actualizados mis datos personales.		

Tabla 5.28. Crear pedido para mesa.

Historia de usuario			
Número	HU14	**Usuario**	Mesero
Nombre	Crear pedido para mesa		
Prioridad	Alta	**Puntos Estimados**	3
Responsable	Katherin Pico	**Iteración Asignada**	Iteración 4
Descripción	Como mesero, quiero crear un pedido asignándolo a una mesa, para registrar el consumo de los clientes en el restaurante.		

Tabla 5.29. Agregar productos al pedido.

Historia de usuario			
Número	HU15	**Usuario**	Mesero
Nombre	Agregar productos al pedido		
Prioridad	Alta	**Puntos Estimados**	3
Responsable	Katherin Pico	**Iteración Asignada**	Iteración 4
Descripción	Como mesero, quiero agregar productos al pedido indicando la cantidad, para registrar correctamente lo que solicita el cliente.		

Tabla 5.30. Editar pedido.

Historia de usuario			
Número	HU16	**Usuario**	Mesero
Nombre	Editar pedido		
Prioridad	Alta	**Puntos Estimados**	3
Responsable	Katherin Pico	**Iteración Asignada**	Iteración 4
Descripción	Como mesero, quiero editar o quitar productos del pedido antes de que lo preparen, para corregir errores.		

Tabla 5.31. Eliminar pedido.

Historia de usuario			
Número	HU17	**Usuario**	Mesero
Nombre	Eliminar pedido		
Prioridad	Media	**Puntos Estimados**	2
Responsable	Katherin Pico	**Iteración Asignada**	Iteración 4
Descripción	Como mesero, quiero eliminar un pedido que aún no se prepara, para evitar errores.		

Tabla 5.32. Ver estado del pedido.

Historia de usuario			
Número	HU18	**Usuario**	Mesero
Nombre	Ver estado del pedido		
Prioridad	Alta	**Puntos Estimados**	2
Responsable	Katherin Pico	**Iteración Asignada**	Iteración 4
Descripción	Como mesero, quiero ver el estado del pedido en tiempo real, para informar al cliente.		

Tabla 5.33. Enviar pedido a cocina.

Historia de usuario			
Número	HU19	**Usuario**	Mesero
Nombre	Enviar pedido a cocina		
Prioridad	Alta	**Puntos Estimados**	2
Responsable	Katherin Pico	**Iteración Asignada**	Iteración 4
Descripción	Como mesero, quiero enviar el pedido a cocina, para que el cocinero empiece a preparar la orden sin retrasos.		

Tabla 5.34. Visualizar pedidos pendientes.

Historia de usuario			
Número	HU20	**Usuario**	Cocinero
Nombre	Visualizar pedidos pendientes		
Prioridad	Alta	**Puntos Estimados**	2
Responsable	Katherin Pico	**Iteración Asignada**	Iteración 6
Descripción	Como cocinero, quiero visualizar los pedidos pendientes del día, para saber qué platos debo preparar y en qué orden.		

Tabla 5.35. Marcar pedido en preparación.

Historia de usuario			
Número	HU21	**Usuario**	Cocinero
Nombre	Marcar pedido en preparación		
Prioridad	Alta	**Puntos Estimados**	2
Responsable	Katherin Pico	**Iteración Asignada**	Iteración 6
Descripción	Como cocinero, quiero marcar un pedido en preparación, para evitar que se realice dos veces el mismo pedido.		

Tabla 5.36. Marcar pedido como listo.

Historia de usuario			
Número	HU22	**Usuario**	Cocinero
Nombre	Marcar pedido como listo		
Prioridad	Alta	**Puntos Estimados**	2
Responsable	Katherin Pico	**Iteración Asignada**	Iteración 6
Descripción	Como cocinero, quiero marcar un pedido como listo, para avisar al mesero o cajero que ya puede ser entregado.		

Tabla 5.37. Avisar producto agotado.

Historia de usuario			
Número	HU23	**Usuario**	Cocinero
Nombre	Avisar producto agotado		
Prioridad	Media	**Puntos Estimados**	3
Responsable	Katherin Pico	**Iteración Asignada**	Iteración 6
Descripción	Como cocinero, quiero avisar cuando un producto se ha agotado, para que no se siga ofreciendo a los clientes.		

Tabla 5.38. Editar perfil cocinero.

Historia de usuario			
Número	HU24	**Usuario**	Cocinero
Nombre	Editar perfil cocinero		
Prioridad	Media	**Puntos Estimados**	3
Responsable	Katherin Pico	**Iteración Asignada**	Iteración 9
Descripción	Como cocinero, quiero editar mi perfil, para mantener actualizada mi información personal.		

Tabla 5.39. Editar perfil cajero.

Historia de usuario			
Número	HU25	**Usuario**	Cajero
Nombre	Editar perfil cajero		
Prioridad	Media	**Puntos Estimados**	3
Responsable	Katherin Pico	**Iteración Asignada**	Iteración 9
Descripción	Como cajero, quiero editar mi perfil, para mantener actualizada mi información personal.		

Tabla 5.40. Registrar pedido a domicilio.

Historia de usuario			
Número	HU26	**Usuario**	Cajero
Nombre	Registrar pedido a domicilio		
Prioridad	Alta	**Puntos Estimados**	3
Responsable	Katherin Pico	**Iteración Asignada**	Iteración 5
Descripción	Como cajero, quiero registrar pedidos a domicilio con los datos del cliente, para asegurar que el pedido llegue correctamente.		

Tabla 5.41. Registrar pago del pedido del restaurante.

Historia de usuario			
Número	HU27	**Usuario**	Cajero
Nombre	Registrar pago del pedido del restaurante		
Prioridad	Alta	**Puntos Estimados**	3
Responsable	Katherin Pico	**Iteración Asignada**	Iteración 7
Descripción	Como cajero, quiero registrar el pago de los pedidos del restaurante, para cerrar la venta y dejar constancia del cobro.		

Tabla 5.42. Ver total del pedido.

Historia de usuario			
Número	HU28	**Usuario**	Cajero
Nombre	Ver total del pedido		
Prioridad	Alta	**Puntos Estimados**	2
Responsable	Katherin Pico	**Iteración Asignada**	Iteración 7
Descripción	Como cajero, quiero ver el total del pedido calculado automáticamente, para cobrar correctamente.		

Tabla 5.43. Aplicar recargo por servicio.

Historia de usuario			
Número	HU29	**Usuario**	Cajero
Nombre	Aplicar recargo por servicio		
Prioridad	Media	**Puntos Estimados**	2
Responsable	Katherin Pico	**Iteración Asignada**	Iteración 7
Descripción	Como cajero, quiero aplicar un recargo cuando el pedido es a domicilio, para cobrar correctamente.		

Tabla 5.44. Generar comprobante de pago.

Historia de usuario			
Número	HU30	**Usuario**	Cajero
Nombre	Generar comprobante de pago		
Prioridad	Media	**Puntos Estimados**	3
Responsable	Katherin Pico	**Iteración Asignada**	Iteración 7
Descripción	Como cajero, quiero generar un comprobante de pago, para entregar un respaldo al cliente.		

Tabla 5.45. HU: Iniciar sesión.

Historia de usuario			
Número	HU31	**Usuario**	Personal del restaurante
Nombre	Iniciar sesión		
Prioridad	Alta	**Puntos Estimados**	1
Responsable	Katherin Pico	**Iteración Asignada**	Iteración 1
Descripción	Como trabajador del restaurante, quiero iniciar sesión en el sistema, para acceder a las funciones según mi rol.		

Tabla 5.46. HU: Cerrar sesión.

Historia de usuario			
Número	HU32	**Usuario**	Personal del restaurante
Nombre	Cerrar sesión		
Prioridad	Media	**Puntos Estimados**	1
Responsable	Katherin Pico	**Iteración Asignada**	Iteración 1
Descripción	Como trabajador del restaurante, quiero cerrar sesión al finalizar mi turno, para evitar que otra persona use mi cuenta.		

Tabla 5.47. Recibir notificaciones.

Historia de usuario			
Número	HU33	**Usuario**	Personal del restaurante
Nombre	Recibir notificaciones		
Prioridad	Alta	**Puntos Estimados**	2
Responsable	Katherin Pico	**Iteración Asignada**	Iteración 9
Descripción	Como trabajador del restaurante, quiero recibir notificaciones del sistema, para actuar rápidamente.		

Tabla 5.48. Uso en dispositivos móviles.

Historia de usuario			
Número	HU34	**Usuario**	Personal del restaurante
Nombre	Uso en dispositivos móviles		
Prioridad	Media	**Puntos Estimados**	3
Responsable	Katherin Pico	**Iteración Asignada**	Iteración 9

Descripción	Como trabajador del restaurante, quiero usar el sistema en dispositivos móviles, para mayor comodidad.

Tabla 5.49. Visualizar pedidos a domicilio.

Historia de usuario			
Número	HU35	**Usuario**	Cajero
Nombre	Visualizar pedidos a domicilio		
Prioridad	Alta	**Puntos Estimados**	2
Responsable	Katherin Pico	**Iteración Asignada**	Iteración 5
Descripción	Como cajero, quiero visualizar los pedidos a domicilio registrados, para gestionar correctamente los pedidos.		

Tabla 5.50. Agregar productos a pedido a domicilio.

Historia de usuario			
Número	HU36	**Usuario**	Cajero
Nombre	Agregar productos a pedido a domicilio		
Prioridad	Alta	**Puntos Estimados**	3
Responsable	Katherin Pico	**Iteración Asignada**	Iteración 5
Descripción	Como cajero, quiero agregar productos al pedido a domicilio, para completar correctamente la orden solicitada por el cliente.		

Tabla 5.51. Editar pedidos a domicilio.

Historia de usuario			
Número	HU37	**Usuario**	Cajero
Nombre	Editar pedidos a domicilio		
Prioridad	Alta	**Puntos Estimados**	3
Responsable	Katherin Pico	**Iteración Asignada**	Iteración 5

Descripción	Como cajero, quiero editar los pedidos a domicilio antes de que los preparen, para corregir errores o actualizar la solicitud del cliente.

Tabla 5.52. Eliminar pedidos a domicilio.

Historia de usuario			
Número	HU38	**Usuario**	Cajero
Nombre	Eliminar pedidos a domicilio		
Prioridad	Media	**Puntos Estimados**	2
Responsable	Katherin Pico	**Iteración Asignada**	Iteración 5
Descripción	Como cajero, quiero eliminar pedidos a domicilio que no hayan sido preparados, para evitar pedidos innecesarios o erróneos.		

Tabla 5.53. Enviar pedidos a domicilio a cocina.

Historia de usuario			
Número	HU39	**Usuario**	Cajero
Nombre	Enviar pedidos a domicilio a cocina		
Prioridad	Alta	**Puntos Estimados**	2
Responsable	Katherin Pico	**Iteración Asignada**	Iteración 5
Descripción	Como cajero, quiero enviar los pedidos a domicilio a cocina, para que sean preparados oportunamente.		

Tabla 5.54. Visualizar estado de pedidos a domicilio.

Historia de usuario			
Número	HU40	**Usuario**	Cajero
Nombre	Visualizar estado de pedidos a domicilio		
Prioridad	Alta	**Puntos Estimados**	2

Responsable	Katherin Pico	**Iteración Asignada**	Iteración 5
Descripción	Como cajero, quiero visualizar el estado de los pedidos a domicilio en tiempo real, para informar al cliente y coordinar el despacho.		

Tabla 5.55. Registrar pago de pedidos a domicilio.

Historia de usuario			
Número	HU41	**Usuario**	Cajero
Nombre	Registrar pago de pedidos a domicilio		
Prioridad	Alta	**Puntos Estimados**	2
Responsable	Katherin Pico	**Iteración Asignada**	Iteración 7
Descripción	Como cajero, quiero registrar el pago de los pedidos a domicilio para completar el proceso de venta y generar el comprobante correspondiente.		

Tabla 5.56. Visualizar reporte unificado para cierre de caja.

Historia de usuario			
Número	HU42	**Usuario**	Cajero
Nombre	Visualizar reporte unificado para cierre de caja		
Prioridad	Media	**Puntos Estimados**	3
Responsable	Katherin Pico	**Iteración Asignada**	Iteración 8
Descripción	Como cajero, quiero visualizar un reporte unificado de los cobros del restaurante y pedidos a domicilio, para realizar correctamente el cierre de caja.		

Tabla 5.57. Visualizar mesas.

Historia de usuario			
Número	HU43	**Usuario**	Administrador

Nombre	Visualizar mesas		
Prioridad	Media	**Puntos Estimados**	2
Responsable	Katherin Pico	**Iteración Asignada**	Iteración 2
Descripción	Como administrador, quiero visualizar las mesas registradas, para gestionar la distribución del restaurante.		

Tabla 5.58. Editar mesas.

Historia de usuario			
Número	HU44	**Usuario**	Administrador
Nombre	Editar mesas		
Prioridad	Media	**Puntos Estimados**	2
Responsable	Katherin Pico	**Iteración Asignada**	Iteración 2
Descripción	Como administrador, quiero editar la información de las mesas, para mantener actualizada su disponibilidad y características.		

Tabla 5.59. Eliminar mesas.

Historia de usuario			
Número	HU45	**Usuario**	Administrador
Nombre	Eliminar mesas		
Prioridad	Baja	**Puntos Estimados**	2
Responsable	Katherin Pico	**Iteración Asignada**	Iteración 2
Descripción	Como administrador, quiero eliminar mesas que no estén en uso, para mantener organizada la gestión del restaurante.		

Tabla 5.60. Registrarse en el sistema.

Historia de usuario			
Número	HU46	**Usuario**	Cliente

Nombre	Registrarse en el sistema		
Prioridad	Alta	**Puntos Estimados**	3
Responsable	Katherin Pico	**Iteración Asignada**	Iteración 10
Descripción	Como cliente, quiero registrarme en el sistema ingresando mis datos personales, para poder realizar pedidos en línea.		

Tabla 5.61. Editar perfil.

Historia de usuario			
Número	HU47	**Usuario**	Cliente
Nombre	Editar perfil		
Prioridad	Media	**Puntos Estimados**	2
Responsable	Katherin Pico	**Iteración Asignada**	Iteración 10
Descripción	Como cliente, quiero editar mi información personal, para mantener actualizados mis datos de contacto.		

Tabla 5.62. Crear pedido.

Historia de usuario			
Número	HU48	**Usuario**	Cliente
Nombre	Crear pedido		
Prioridad	Alta	**Puntos Estimados**	3
Responsable	Katherin Pico	**Iteración Asignada**	Iteración 10
Descripción	Como cliente, quiero crear un pedido desde la plataforma, para solicitar productos del restaurante sin necesidad de llamar.		

Tabla 5.63. Agregar productos al pedido.

Historia de usuario			
Número	HU49	**Usuario**	Cliente

Nombre	Agregar productos al pedido		
Prioridad	Alta	**Puntos Estimados**	3
Responsable	Katherin Pico	**Iteración Asignada**	Iteración 10
Descripción	Como cliente, quiero agregar productos al pedido indicando cantidad, para personalizar mi orden.		

Tabla 5.64. Subir comprobante de pago.

Historia de usuario			
Número	HU50	**Usuario**	Cliente
Nombre	Subir comprobante de pago		
Prioridad	Alta	**Puntos Estimados**	3
Responsable	Katherin Pico	**Iteración Asignada**	Iteración 10
Descripción	Como cliente, quiero subir el comprobante de transferencia bancaria, para validar el pago de mi pedido.		

Tabla 5.65. Enviar pedido para validación.

Historia de usuario			
Número	HU51	**Usuario**	Cliente
Nombre	Enviar pedido para validación		
Prioridad	Alta	**Puntos Estimados**	2
Responsable	Katherin Pico	**Iteración Asignada**	Iteración 10
Descripción	Como cliente, quiero enviar mi pedido al cajero después de subir el comprobante, para que sea validado y procesado.		

Tabla 5.66. Validar comprobante de pago.

Historia de usuario			
Número	HU52	**Usuario**	Cajero

Nombre	Validar comprobante de pago		
Prioridad	Alta	**Puntos Estimados**	3
Responsable	Katherin Pico	**Iteración Asignada**	Iteración 11
Descripción	Como cajero, quiero revisar el comprobante de transferencia enviado por el cliente, para verificar que el pago sea correcto.		

Tabla 5.67. Aprobar pedido en línea.

Historia de usuario			
Número	HU53	**Usuario**	Cajero
Nombre	Aprobar pedido en línea		
Prioridad	Alta	**Puntos Estimados**	2
Responsable	Katherin Pico	**Iteración Asignada**	Iteración 11
Descripción	Como cajero, quiero aprobar el pedido cuando el pago es válido, para enviarlo automáticamente a cocina.		

Tabla 5.68. Rechazar pedido en línea.

Historia de usuario			
Número	HU54	**Usuario**	Cajero
Nombre	Rechazar pedido en línea		
Prioridad	Alta	**Puntos Estimados**	2
Responsable	Katherin Pico	**Iteración Asignada**	Iteración 11
Descripción	Como cajero, quiero rechazar el pedido cuando el pago no sea correcto, para notificar al cliente que su pedido fue cancelado.		

5.3.3 Casos de uso

En la figura 5.13 se visualiza el caso de uso general de todo el sistema.

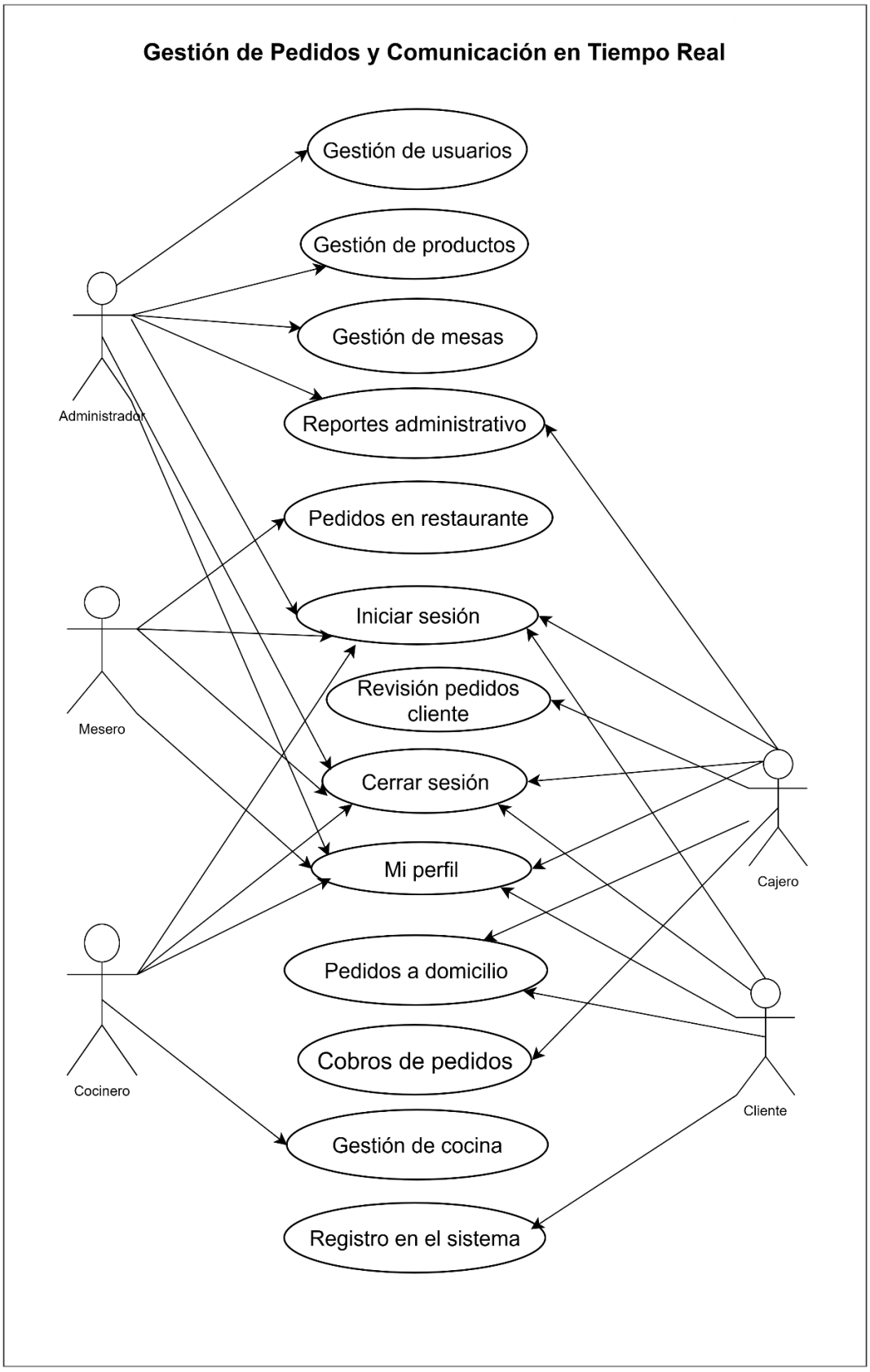


Figura 5.13. Caso de uso: Gestión de pedidos y comunicación en tiempo real.

En la figura 5.14 se observa el caso de uso de la gestión de productos por parte del administrador.

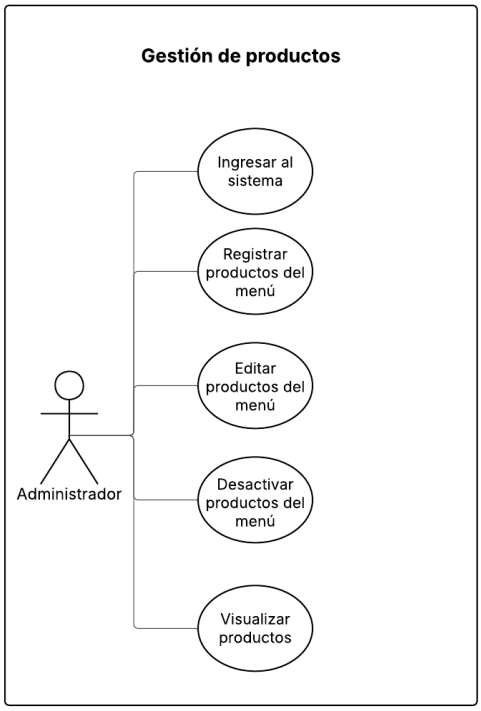


Figura 5.14. Caso de uso: Gestión de productos.

En la figura 5.15 se visualiza el caso de uso de gestión reportes administrativos por parte del administrador.

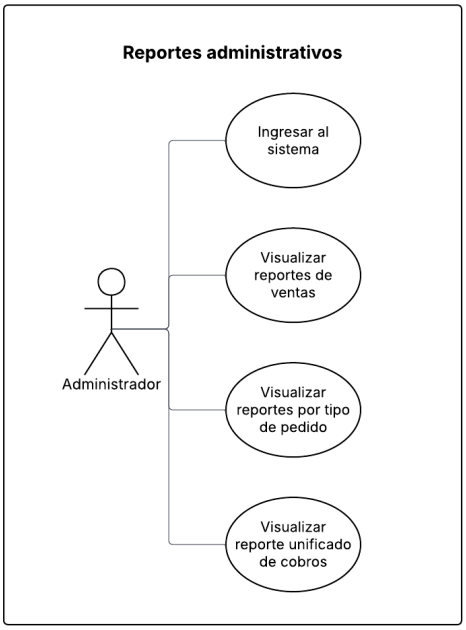


Figura 5.15. Caso de uso: Reportes administrativos.

En la figura 5.16 se observa el caso de uso gestión de usuarios por parte del administrador.

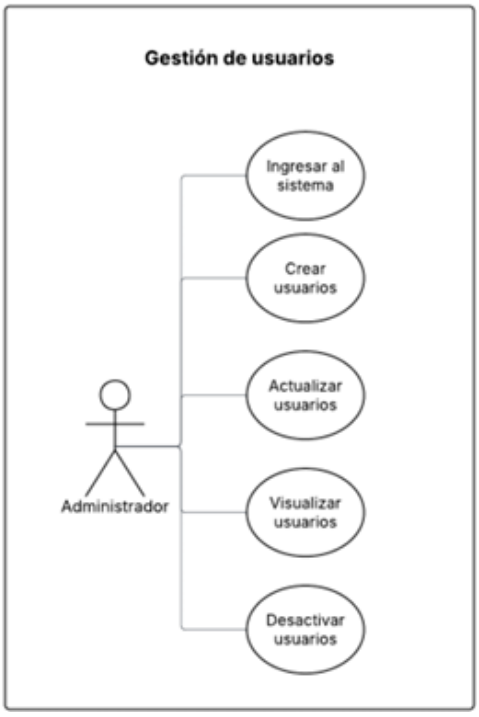


Figura 5.16. Caso de uso: Gestión de usuarios.

En la figura 5.17 se observa el caso de uso gestión de mesas por parte del administrador.

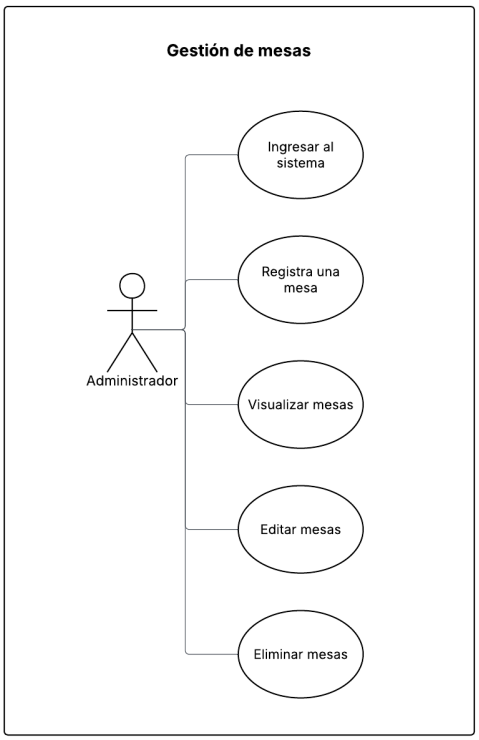


Figura 5.17. Caso de uso: Gestión de mesas

En la figura 5.18 se observa el caso de uso perfil del administrador.

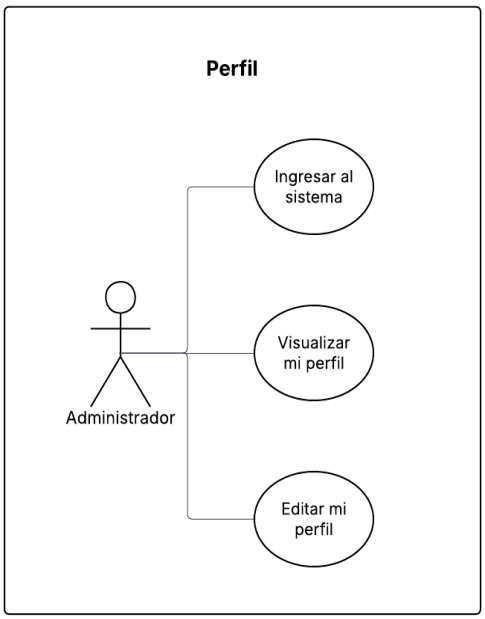


Figura 5.18. Caso de uso: Perfil del administrador.

En la figura 5.19 se observa el caso de uso pedidos en restaurante por parte del mesero.

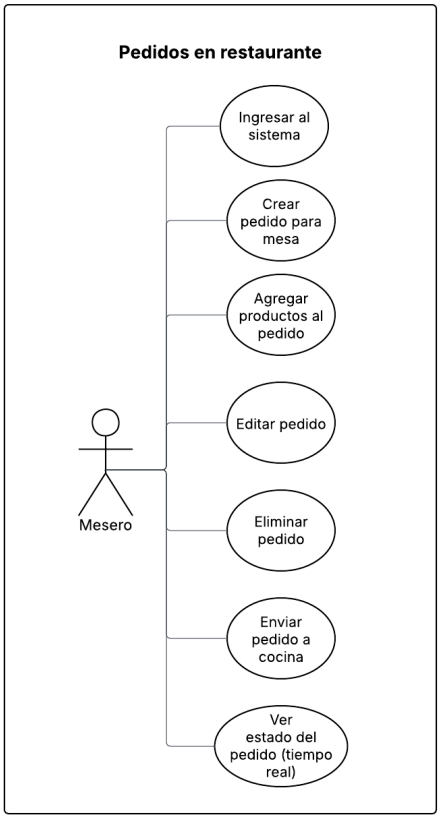


Figura 5.19. Caso de uso: Pedidos en restaurante.

En la figura 5.20 se observa el caso de uso perfil del mesero.

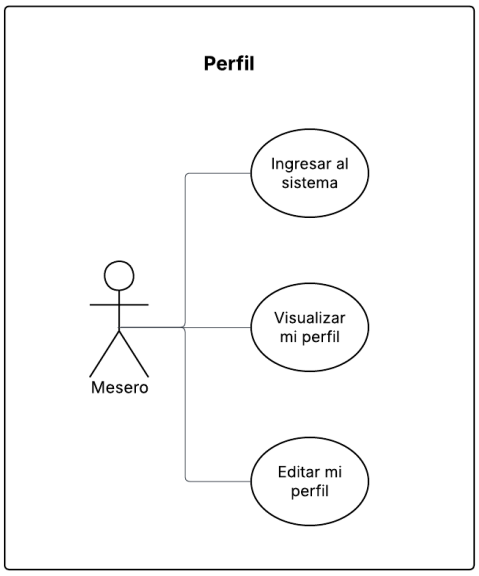


Figura 5.20. Caso de uso: Perfil del mesero.

En la figura 5.21 se observa el caso de uso de pedidos a domicilio por parte del cajero.

DIAGRAMA CASO DE USO DEL CAJERO

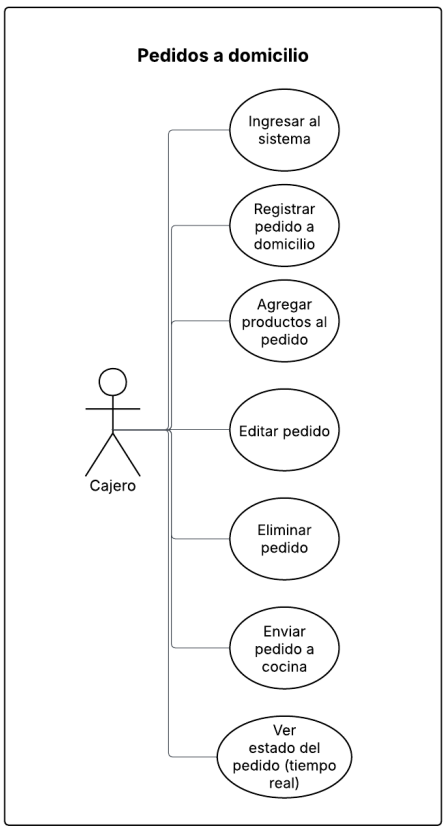


Figura 5.21. Caso de uso Pedidos a domicilio.

En la figura 5.22 de observa el caso de uso revisión pedidos cliente por parte del cajero.

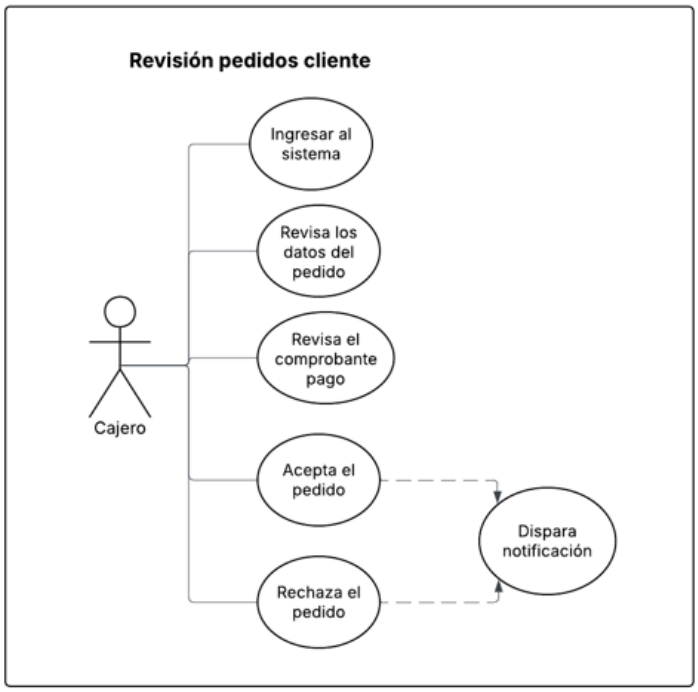


Figura 5.22. Caso de uso: Revisión pedidos cliente.

En la figura 5.23 se observa el caso de uso cobros de pedidos por parte del cajero.

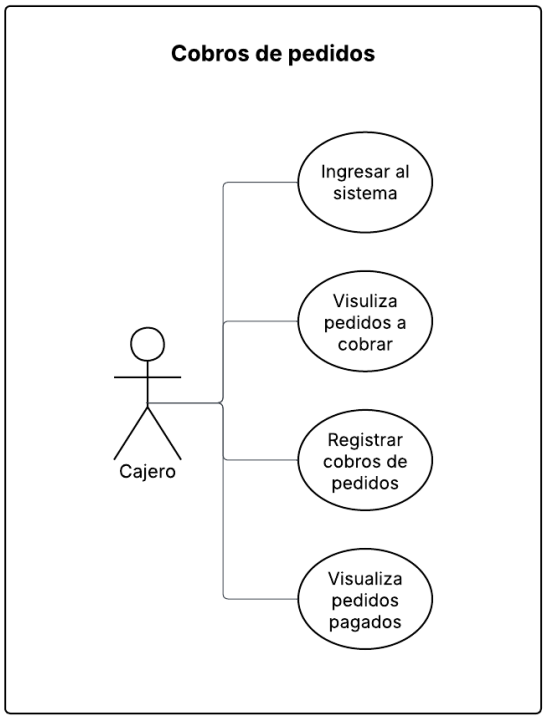


Figura 5.23. Caso de uso: Cobros de pedidos.

En la figura 5.24 se visualiza el caso de uso reporte por parte del cajero.

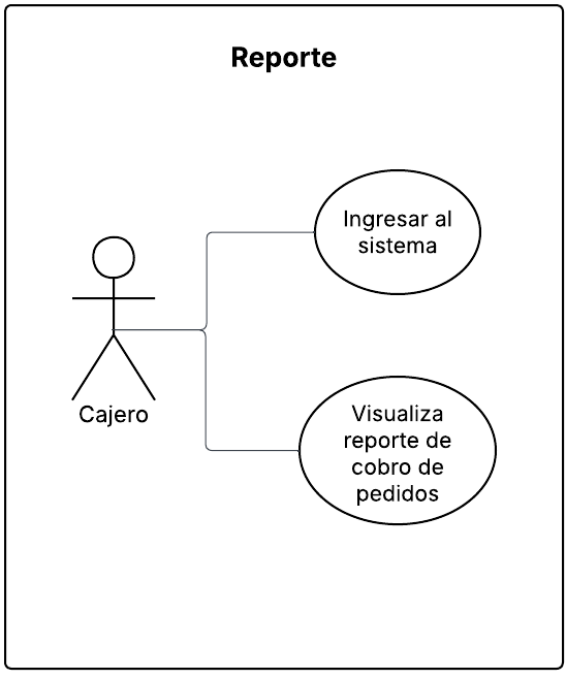


Figura 5.24. Caso de uso: Reporte.

En la figura 5.25 se visualiza el caso de uso perfil del cajero.

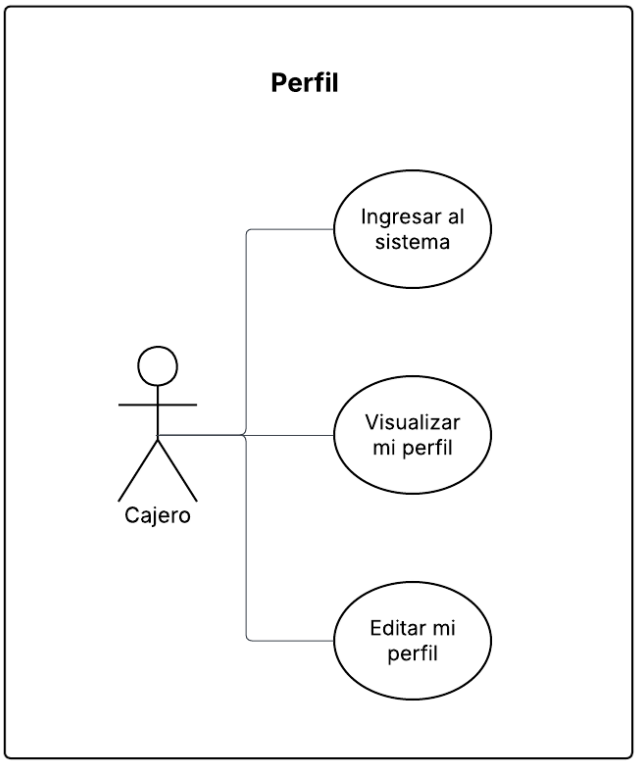


Figura 5.25. Caso de uso: Perfil del cajero.

En la figura 5.26 se observa el caso de uso gestión de cocina por parte de los cocineros.

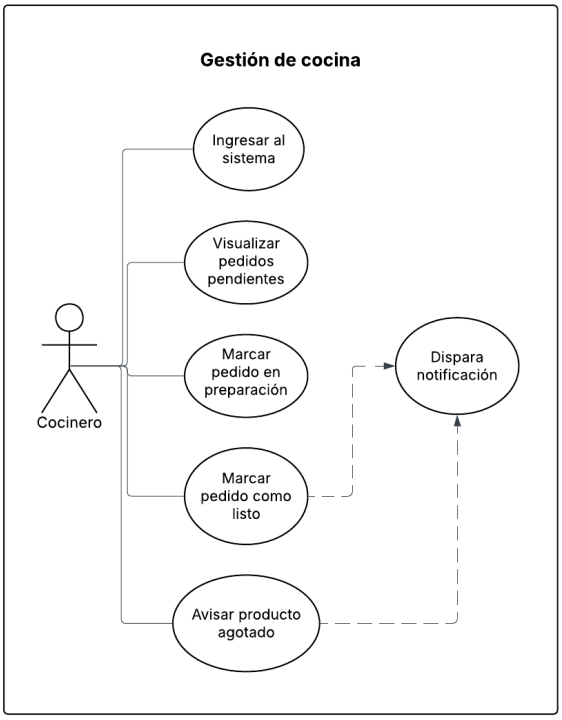


Figura 5.26. Caso de uso: Gestión de cocina.

En la figura 5.27 se observa el caso de uso perfil del cocinero.

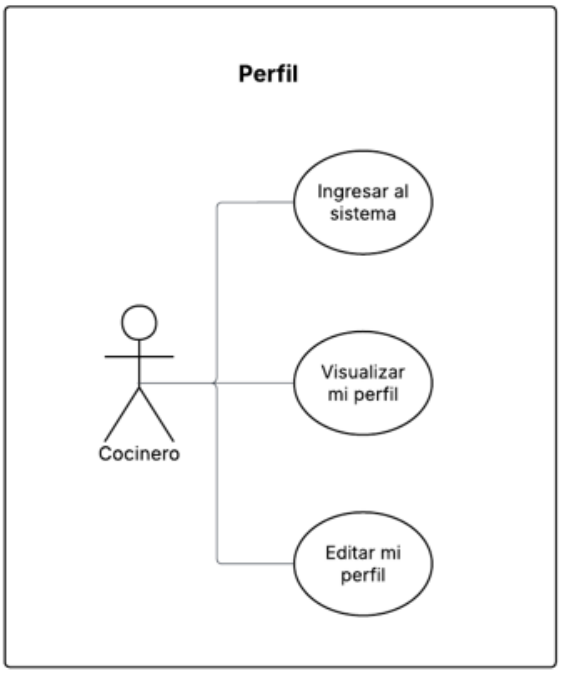


Figura 5.27. Caso de uso: Perfil del cocinero.

En la figura 5.28 se observa el caso de registró en el sistema por parte del cliente.

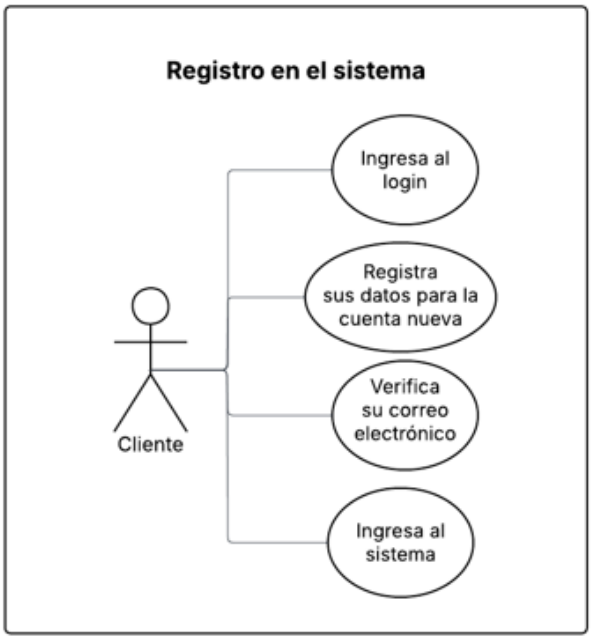


Figura 5.28. Caso de uso: Registro en el sistema.

En la figura 5.29 se observa el caso de uso pedidos a domicilio realizados por los clientes.

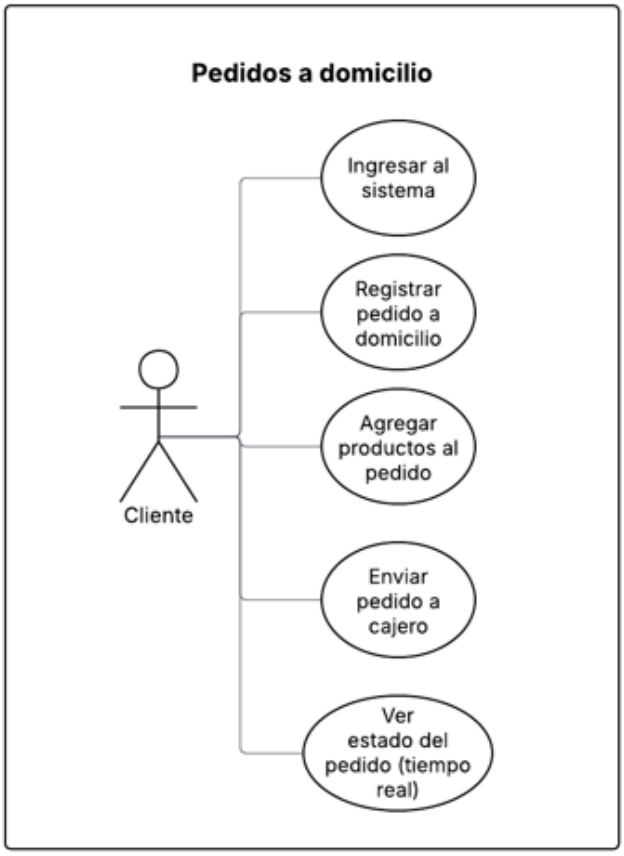


Figura 5.29. Caso de uso: Pedidos a domicilio.

En la figura 5.30 se observa el caso de uso perfil del cliente.

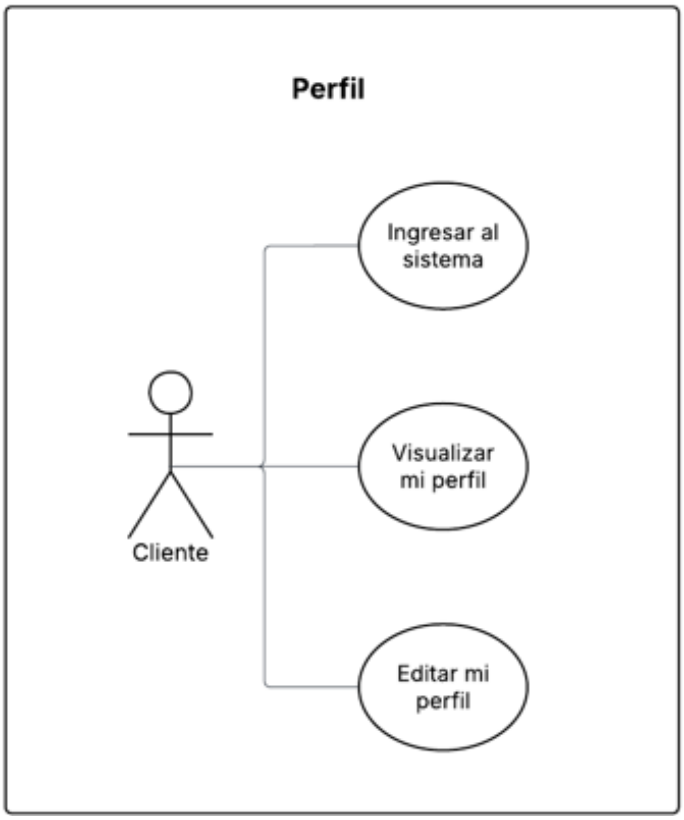


Figura 5.30. Caso de uso Perfil del cliente.

5.3.4 Release Planning

El Release Planning (planificación de entregas) con la estimación de puntos permitió organizar las historias de usuario en entregas progresivas, considerando tanto la prioridad funcional como el esfuerzo requerido para su desarrollo. En este proceso, cada historia de usuario se evaluó tanto en su complejidad como en el tiempo necesario para su implementación, para así estimar en puntos, lo cual permitió distribuirlas de manera equilibrada dentro de cada iteración, asegurando una carga de trabajo adecuada.

A continuación, se mostrará el modelo de Release Planning que se usará para planificar cada iteración.

Tabla 5.69. Release Planning detallado.

Iteración: Este apartado identifica el número de la iteración en la que se agrupan un conjunto de historias de usuario.			
Número	**Historia de Usuario**	**Prioridad**	**Puntos**
Corresponde al identificador de la historia de usuario, el cual permite reconocerla de forma única dentro del proyecto.	En este apartado se presenta el nombre de la historia de usuario que será desarrollada en la iteración.	La prioridad indica el nivel de importancia de la historia de usuario dentro del sistema.	Este campo representa la estimación del esfuerzo necesario para desarrollar la historia de usuario.
Total de puntos	El total de puntos corresponde a la suma de los puntos estimados de todas las historias de usuario incluidas en una iteración.		

En base a la Tabla 5.69. correspondiente al Release Planning detallado, se describen a continuación cada una de las iteraciones con sus respectivas historias de usuario que se obtuvieron en las reuniones con los usuarios finales.

Tabla 5.70. Iteración 1 – Gestión de acceso al sistema.

Iteración 1 – Gestión de acceso al sistema			
Número	**Historia de Usuario**	**Prioridad**	**Puntos**
HU01	Crear usuarios	Alta	3
HU02	Actualizar usuarios	Alta	2
HU03	Visualizar usuarios	Alta	2
HU04	Desactivar usuarios	Alta	2
HU31	Iniciar sesión	Alta	1
HU32	Cerrar sesión	Media	1
Total de puntos			11

Objetivo de la iteración: Permitir que el personal del restaurante acceda al sistema de forma segura y que el administrador gestione correctamente a los usuarios.

Tabla 5.71. Iteración 2 – Gestión de mesas del restaurante.

Iteración 2 – Gestión de mesas del restaurante			
Número	**Historia de Usuario**	**Prioridad**	**Puntos**
HU05	Registrar mesas	Alta	3
HU43	Visualizar mesas	Media	2
HU44	Editar mesas	Media	2
HU45	Eliminar mesas	Baja	2
Total de puntos			9

Objetivo de la iteración: Gestionar correctamente las mesas del restaurante para facilitar la atención y organización del servicio.

Tabla 5.72. Iteración 3 – Gestión del menú.

Iteración 3 – Gestión del menú			
Número	**Historia de Usuario**	**Prioridad**	**Puntos**
HU06	Registrar productos del menú	Alta	3
HU08	Editar productos del menú	Alta	3
HU09	Desactivar productos del menú	Media	2
Total de puntos			8

Objetivo de la iteración: Administrar el menú del restaurante asegurando la correcta disponibilidad y actualización de los productos.

Tabla 5.73. Iteración 4 – Gestión de pedidos en restaurante.

Iteración 4 – Gestión de pedidos en restaurante			
Número	**Historia de Usuario**	**Prioridad**	**Puntos**
HU14	Crear pedido para mesa	Alta	3
HU15	Agregar productos al pedido	Alta	3
HU16	Editar pedido	Alta	3
HU17	Eliminar pedido	Media	2
HU18	Ver estado del pedido	Alta	2
HU19	Enviar pedido a cocina	Alta	2
Total de puntos			15

Objetivo de la iteración: Permitir al mesero gestionar pedidos del restaurante de forma eficiente y en tiempo real.

Tabla 5.74. Iteración 5 – Gestión de pedidos a domicilio.

Iteración 5 – Gestión de pedidos a domicilio			
Número	**Historia de Usuario**	**Prioridad**	**Puntos**
HU26	Registrar pedido a domicilio	Alta	3
HU35	Visualizar pedidos a domicilio	Alta	2
HU36	Agregar productos a pedido a domicilio	Alta	3
HU37	Editar pedidos a domicilio	Alta	3
HU38	Eliminar pedidos a domicilio	Media	2
HU39	Enviar pedidos a domicilio a cocina	Alta	2
HU40	Visualizar estado de pedidos a domicilio	Alta	2
Total de puntos			17

Objetivo de la iteración: Gestionar pedidos a domicilio desde su registro hasta el envío a cocina y seguimiento del estado.

Tabla 5.75. Iteración 6 – Gestión de cocina.

Iteración 6 – Gestión de cocina			
Número	**Historia de Usuario**	**Prioridad**	**Puntos**
HU20	Visualizar pedidos pendientes	Alta	2
HU21	Marcar pedido en preparación	Alta	2
HU22	Marcar pedido como listo	Alta	2
HU23	Avisar producto agotado	Media	3
Total de puntos			9

Objetivo de la iteración: Facilitar el control de pedidos en cocina y la comunicación con el personal del restaurante.

Tabla 5.76. Iteración 7 – Gestión de pagos y cobros.

Iteración 7 – Gestión de pagos y cobros			
Número	**Historia de Usuario**	**Prioridad**	**Puntos**
HU27	Registrar pago del pedido del restaurante	Alta	3
HU28	Ver total del pedido	Alta	2
HU29	Aplicar recargo por servicio	Media	2
HU30	Generar comprobante de pago	Media	3
HU41	Registrar pago de pedidos a domicilio	Alta	2
Total de puntos			12

Objetivo de la iteración: Gestionar correctamente los pagos de pedidos del restaurante y a domicilio, generando comprobantes de pago.

Tabla 5.77. Iteración 8 – Reportes y control administrativo.

Iteración 8 – Reportes y control administrativo			
Número	**Historia de Usuario**	**Prioridad**	**Puntos**
HU07	Visualizar reportes de ventas	Media	3
HU10	Visualizar reportes por tipo de pedido	Media	3
HU11	Visualizar reporte unificado de cobros	Media	3
HU42	Visualizar reporte unificado para cierre de caja	Media	3
Total de puntos			12

Objetivo de la iteración: Proporcionar reportes administrativos que apoyen la toma de decisiones y el control financiero.

Tabla 5.78. Iteración 9 – Perfil, notificaciones y uso móvil.

Iteración 9 – Perfil, notificaciones y uso móvil			
Número	**Historia de Usuario**	**Prioridad**	**Puntos**
HU12	Editar perfil administrador	Media	2
HU13	Editar perfil mesero	Media	2
HU24	Editar perfil cocinero	Media	3
HU25	Editar perfil cajero	Media	3
HU33	Recibir notificaciones	Alta	2
HU34	Uso en dispositivos móviles	Media	3
Total de puntos			15

Objetivo de la iteración: Mejorar la experiencia del usuario mediante personalización del perfil, notificaciones y compatibilidad móvil.

Tabla 5.79. Iteración 10 – Módulo Cliente (Pedidos en línea).

Iteración 10 – Módulo Cliente (Pedidos en línea)			
Número	**Historia de Usuario**	**Prioridad**	**Puntos**
HU46	Registro de cliente	Alta	3
HU47	Editar perfil cliente	Media	2
HU48	Crear pedido	Alta	3
HU49	Agregar productos	Alta	3
HU50	Subir comprobante	Alta	3
HU51	Enviar pedido al cajero	Alta	2
Total de puntos			16

Objetivo de la iteración: Permitir que el cliente gestione pedidos en línea incluyendo registro, selección de productos y envío del comprobante de pago.

Tabla 5.80. Iteración 11 – Validación de pagos en línea.

Iteración 11 – Validación de pagos en línea			
Número	**Historia de Usuario**	**Prioridad**	**Puntos**
HU52	Validar comprobante	Alta	3
HU53	Aprobar pedido	Media	2
HU54	Rechazar pedido	Alta	2
Total de puntos			7

Objetivo de la iteración: Permitir al cajero validar pagos de pedidos en línea y automatizar el envío a cocina o cancelación.

Tabla 5.81. Resumen general del Release Planning.

Iteración	Nombre	Total de puntos	Tiempo	Periodo
Iteración 1	Gestión de acceso al sistema	11	1 semana	30/10/2025 – 05/11/2025
Iteración 2	Gestión de mesas del restaurante	9	1 semana	06/11/2025 – 12/11/2025
Iteración 3	Gestión del menú	8	1 semana	13/11/2025 – 19/11/2025
Iteración 4	Gestión de pedidos en restaurante	15	2 semanas	20/11/2025 – 03/12/2025
Iteración 5	Gestión de pedidos a domicilio	17	2 semanas	04/12/2025 – 17/12/2025
Iteración 6	Gestión de cocina	9	1 semana	18/12/2025 – 24/12/2025
Iteración 7	Gestión de pagos y cobros	12	1 semana	25/12/2025 – 31/12/2025
Iteración 8	Reportes y control administrativo	12	1 semana	01/01/2026 – 07/01/2026
Iteración 9	Perfil, notificaciones y uso móvil	15	1 semana	08/01/2026 – 14/01/2026
Iteración 10	Módulo Cliente (Pedidos en línea)	16	2 semanas	15/01/2026 – 28/01/2026
Iteración 11	Validación de pagos en línea	7	1 semanas	29/01/2026 – 04/02/2026

5.3.5 Desarrollo del sistema por iteraciones

A continuación se muestra las iteraciones desarrolladas para la creación de la aplicación web progresiva.

5.3.5.1 Iteración 1 – Gestión de acceso al sistema

5.3.5.1.1 Planificación

En base a las historias de usuario anteriormente realizadas se van a mencionar a continuación las que pertenecen a la iteración 1.

Tabla 5.82. Historias de usuario de la iteración 1.

Número	Historia de Usuario
HU01	Crear usuarios
HU02	Actualizar usuarios
HU03	Visualizar usuarios
HU04	Desactivar usuarios
HU31	Iniciar sesión
HU32	Cerrar sesión

5.3.5.1.2 Diseño

5.3.5.1.2.1 Prototipos de interfases

En la figura 5.31 se observa el panel de inicio de sesión en donde los trabajadores (mesero, administrador y cocineros) ingresaran sus credenciales para poder tener acceso al sistema.

Figura 5.31. Login.

En la figura 5.32 se observa el listado de todos los trabajadores registrados en el sistema por el administrador.

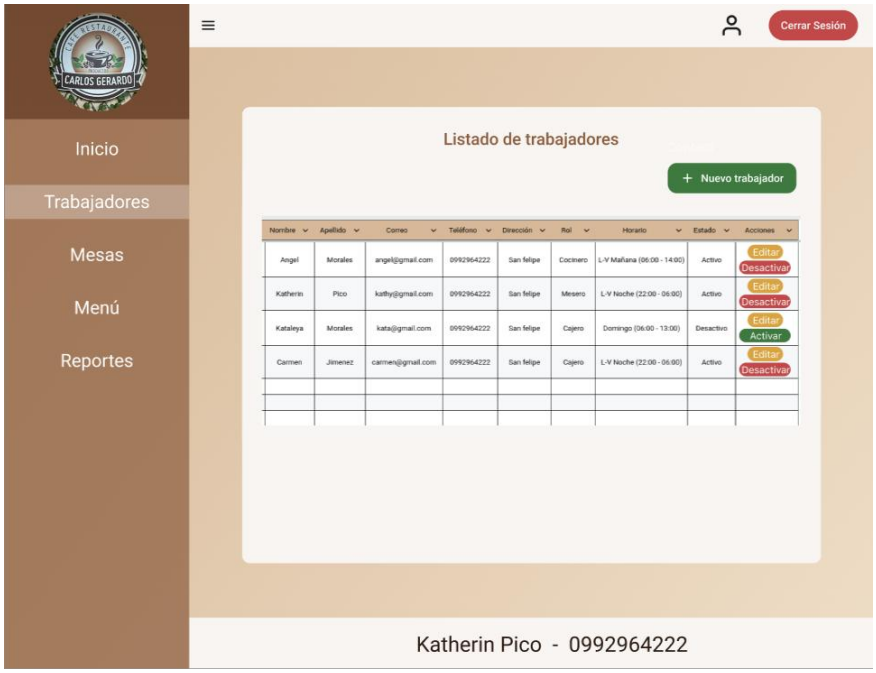


Figura 5.32. Listado de trabajadores.

En la figura 5.33 se observa el formulario de registro de trabajadores en donde se ingresarán sus datos personales y las credenciales con las que ingresara sesión al sistema.

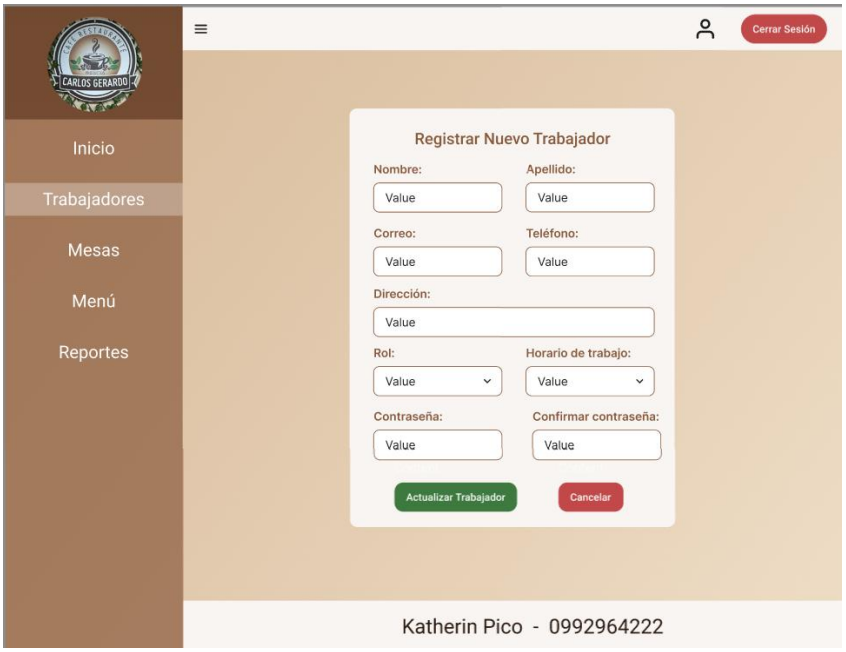


Figura 5.33. Formulario de registro de trabajadores.

En la figura 5.34 se observa el formulario para editar los datos del trabajador ya anteriormente registrados en el sistema; en caso de pedida de contraseña, solo el administrador podrá hacer cambio de contraseña, así como la actualización de los datos personales.

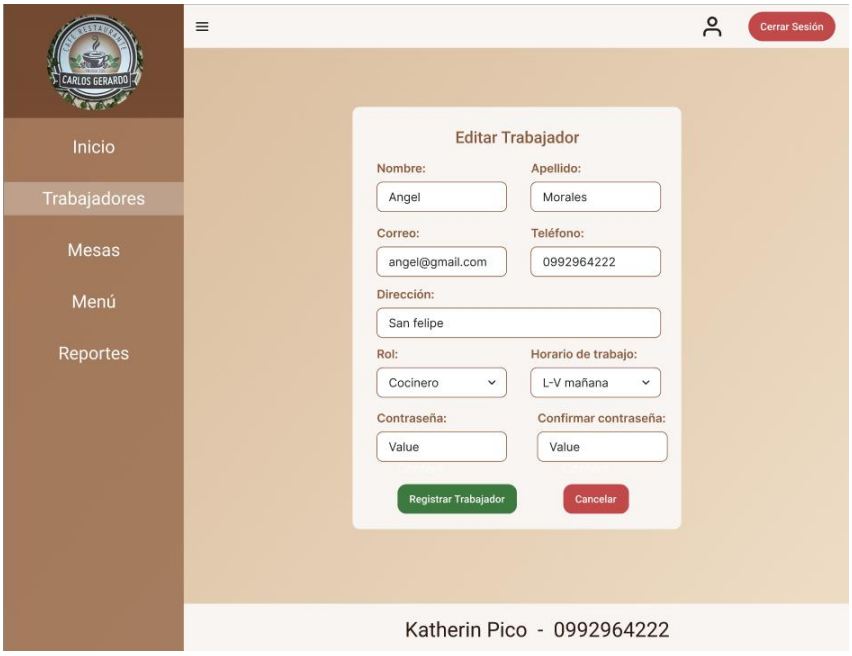


Figura 5.34. Formulario de editar datos del trabajador.

En la figura 5.35 se observa un mensaje de confirmación para desactivar un trabajador específico registrado en el sistema, el cual ya haya realizado registros o interactuado con el sistema; en caso de no haberlo hecho, se permitirá eliminar definitivamente sus datos.

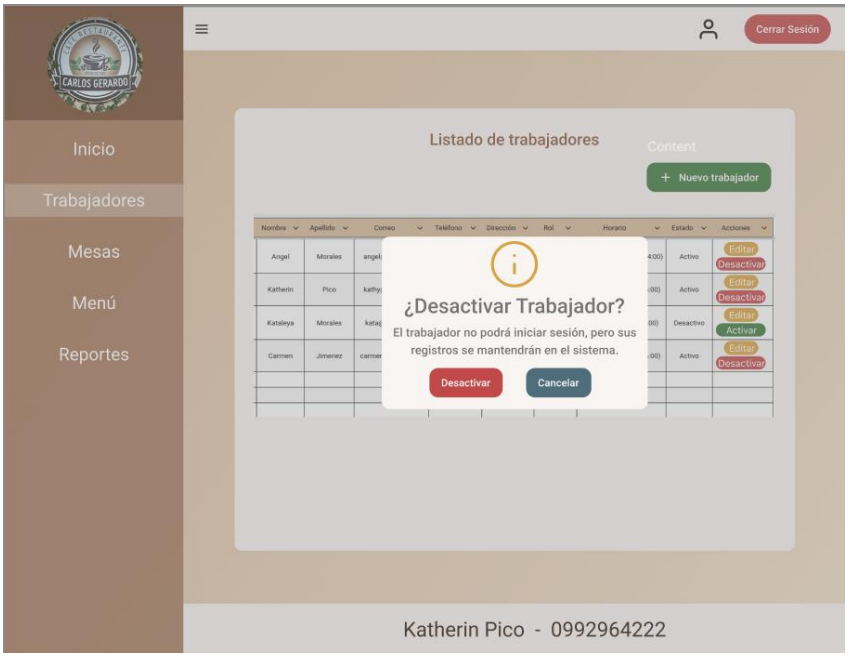


Figura 5.35. Mensaje de confirmación para desactivar un trabajador.

5.3.5.1.2.2 Diseño de bases de datos

En la siguiente figura se presenta el diseño de la base de datos del sistema, en la cual se muestran las tablas que conforman la estructura de almacenamiento de la información, así como, las relaciones existentes entre ellas. Este modelo permite organizar y gestionar de manera eficiente los datos relacionados con los usuarios, pedidos, pagos y demás elementos necesarios para el funcionamiento de la aplicación.

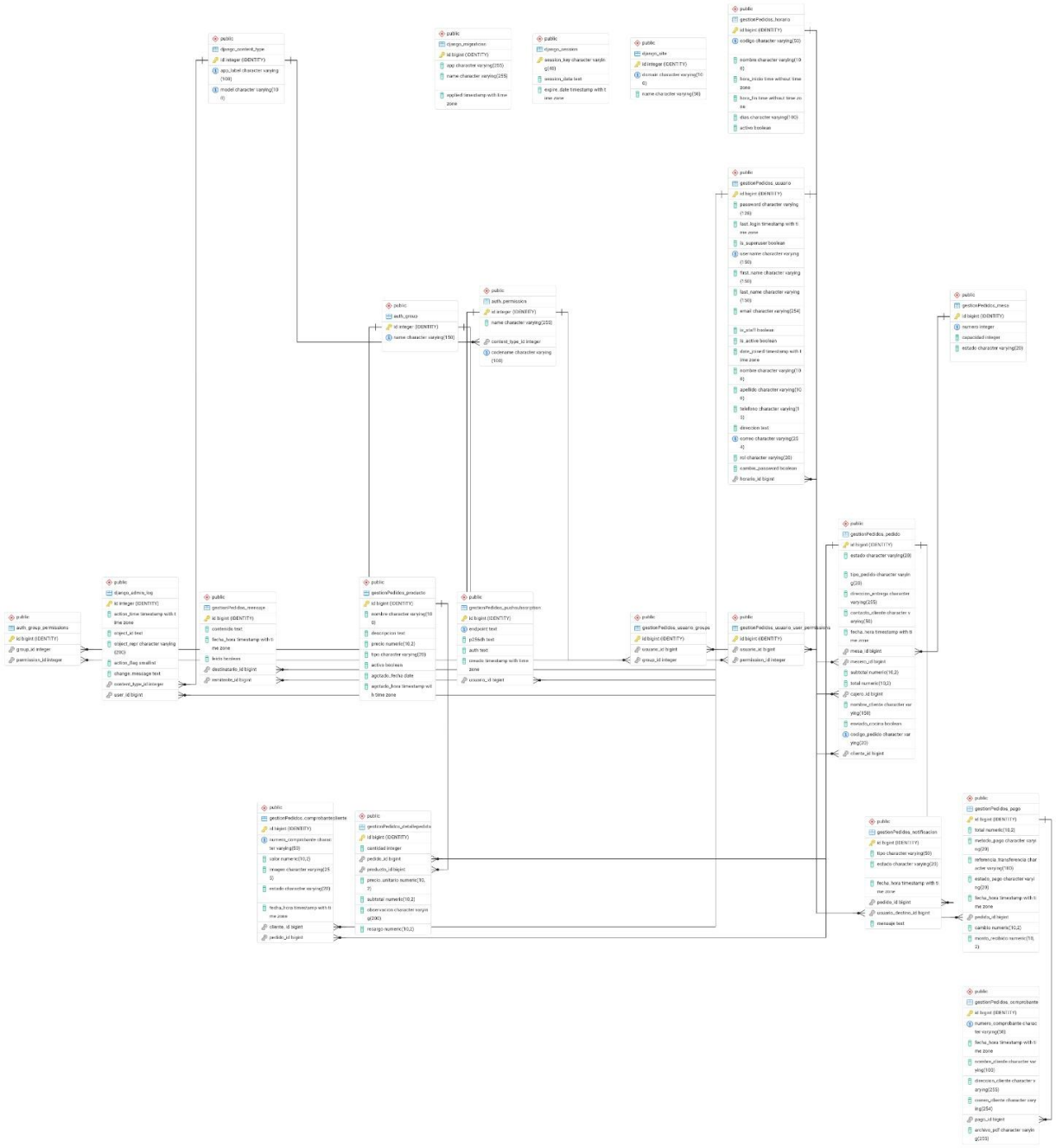


Figura 5.36. Diseño de la base de datos.

5.3.5.1.3 Codificación

En la figura 5.37 se evidencia el código del template en donde se va a mostrar el login para que los usuarios inicien sesión.

```
<html lang="es">
<body>
<div class="container">
    <div class="row justify-content-center">
        <div class="col-sm-12 col-md-10 col-lg-8 col-xl-6">
                    </div>

                    <!-- Formulario -->
                    <div class="col-md-6 bg-white">
                        <div class="card-body p-4">

                            <div class="text-center mb-4">
                                <h2 class="restaurant-title">Bienvenido</h2>
                                <p class="text-muted">Inicia sesión para continuar</p>
                            </div>

                            <form id="frm_login" method="post" action="{% url 'iniciar_sesion' %}">
                                {% csrf_token %}

                               {% if messages %}
                                    {% for message in messages %}
                                        <div class="alert
                                        {% if message.tags == 'error' %} alert-danger
                                        {% elif message.tags == 'success' %} alert-success
                                        {% elif message.tags == 'info' %} alert-info
                                        {% else %} alert-secondary
                                        {% endif %}
                                        py-2" role="alert">
                                        {{ message }}
                                        </div>
                                    {% endfor %}
                                {% endif %}
                                <!-- Correo -->
                                <div class="form-group mb-3">
                                    <label for="correo" class="fw-semibold">Correo electrónico</label>
                                    <div class="input-group">
                                        <span class="input-group-text">
                                            <i class="fa-solid fa-envelope"></i>
                                        </span>
                                        <input id="correo" type="email" name="correo" class="form-control" placeholder="usuario@correo.com" required>
                                    </div>
                                </div>

                                <!-- Contraseña -->
                                <div class="form-group mb-4">
```

Figura 5.37. Template: Login.

En la figura 5.38 se evidencia el código del views en donde se programa la lógica del backend para el login.

```python
def login_view(request):
    return render(request, "login/login.html")

# ----------------------------
# Iniciar sesión
# ----------------------------
def iniciar_sesion(request):
    if request.method == 'POST':
        correo = request.POST.get('correo')
        password = request.POST.get('password')

        usuario = authenticate(request, correo=correo, password=password)

        if usuario is not None:
            auth_login(request, usuario)

            # Guardar datos en sesión
            request.session['usuario_id'] = usuario.id
            request.session['usuario_rol'] = usuario.rol
            request.session['usuario_nombre'] = f"{usuario.nombre} {usuario.apellido}"

            # Si es su primer ingreso, debe cambiar su contraseña
            if not usuario.cambio_password:
                return redirect('cambiar_password_primera_vez')

            # Redirección según rol
            if usuario.rol == 'admin':
                return redirect('dashboard_admin')
            elif usuario.rol == 'mesero':
                return redirect('dashboard_mesero')
            elif usuario.rol == 'cocinero':
                return redirect('vista_cocina')
            elif usuario.rol == 'cajero':
                return redirect('dashboard_cajero')
            else:
                messages.error(request, "Rol desconocido.")
        else:
            messages.error(request, "Correo o contraseña incorrectos.")

    return render(request, 'login/login.html')
```

Figura 5.38. Views: Login

En la figura 5.39 se evidencia el código del template en donde se va a mostrar una lista de los trabajadores registrados en la base de datos.

```
<div class="main-content-wrapper">
    <div class="container-fluid mt-5">
        <div class="content-card">
            <div class="table-responsive">
                <table class="table table-bordered table-striped table-hover" id="tbl_trabajadores">
                    <thead class="table-primary">
                        <tr>
                            <th>Nombre</th>
                            <th>Apellido</th>
                            <th>Correo</th>
                            <th>Teléfono</th>
                            <th>Dirección</th>
                            <th>Rol</th>
                            <th>Horario</th>
                            <th>Estado</th>
                            <th>Acciones</th>
                        </tr>
                    </thead>
                    <tbody>
                        {% for trabajador in trabajadores %}
                        <tr>
                            <td>{{ trabajador.nombre }}</td>
                            <td>{{ trabajador.apellido }}</td>
                            <td>{{ trabajador.correo }}</td>
                            <td>{{ trabajador.telefono|default:"—" }}</td>
                            <td>{{ trabajador.direccion|default:"—" }}</td>
                            <td>
                                {% if trabajador.rol == 'mesero' %}
                                <span class="badge bg-primary">Mesero</span>
                                {% elif trabajador.rol == 'cocinero' %}
                                <span class="badge bg-warning">Cocinero</span>
                                {% elif trabajador.rol == 'cajero' %}
                                <span class="badge bg-success">Cajero</span>
                                {% endif %}
                            </td>

                            <td>
                                {% if trabajador.horario %}
                                    <span class="badge badge-horario">{{ trabajador.get_horario_display }}</span>
                                {% else %}
                                    —
                                {% endif %}
                            </td>

```

Figura 5.39. Template: Lista de trabajadores.

En la figura 5.40 se evidencia el código del models (modelo), en donde se está creando la tabla de usuarios para el registro en la base de datos.

```python
ROL_CHOICES = (
    ('admin', 'Administrador'),
    ('mesero', 'Mesero'),
    ('cocinero', 'Cocinero'),
    ('cajero', 'Cajero'),
)

HORARIO_CHOICES = (
    ('lv_manana', 'L-V Mañana (06:00 - 14:00)'),
    ('lv_tarde',  'L-V Tarde (14:00 - 22:00)'),
    ('lv_noche',  'L-V Noche (22:00 - 06:00)'),

    ('sab_manana', 'Sábado Mañana (06:00 - 12:00)'),
    ('sab_tarde',  'Sábado Tarde (11:00 - 15:00)'),
    ('dom_unico',  'Domingo Único (06:00 - 13:00)'),
)

class Usuario(AbstractUser):
    nombre = models.CharField(max_length=100)
    apellido = models.CharField(max_length=100)
    telefono = models.CharField(max_length=15, blank=True, null=True)
    direccion = models.TextField(blank=True, null=True)
    correo = models.EmailField(unique=True)
    rol = models.CharField(max_length=20, choices=ROL_CHOICES)

    horario = models.CharField(max_length=20, choices=HORARIO_CHOICES, blank=True, null=True)

    cambio_password = models.BooleanField(default=False)

    groups = models.ManyToManyField(
        Group,
        related_name='usuarios',
        blank=True,
        help_text='Grupos a los que pertenece el usuario.',
        verbose_name='grupos',
    )
    user_permissions = models.ManyToManyField(
        Permission,
        related_name='usuarios_permisos',
        blank=True,
        help_text='Permisos específicos para el usuario.',
        verbose_name='permisos de usuario',
    )

    USERNAME_FIELD = 'correo'
```

Figura 5.40. Models: Usuarios del sistema.

En la figura 5.41 se observa el código del views en donde se programa la lógica del backend para el CRUD de los trabajadores registrados por el administrador.

```python
# ----------------------------
# Admin-Gestion de trabajadores
# ----------------------------
@login_required(login_url='login')
@rol_requerido('admin')
def listar_trabajadores(request):
    # Filtrar todos los usuarios que no sean clientes ni admin
    trabajadores = Usuario.objects.exclude(rol__in=['cliente', 'admin'])
    return render(request, 'administrador/trabajadores/trabajadores.html', {'trabajadores': trabajadores})

@login_required(login_url='login')
@rol_requerido('admin')
def crear_trabajador(request):
    TOPES = {
        'mesero': 4,
        'cajero': 3,
        'cocinero': 4,
    }

    if request.method == 'POST':
        nombre = request.POST.get('nombre', '').strip()
        apellido = request.POST.get('apellido', '').strip()
        correo = request.POST.get('correo', '').strip()
        telefono = request.POST.get('telefono', '').strip()
        direccion = request.POST.get('direccion', '').strip()
        rol = request.POST.get('rol', '').strip()
        horario = request.POST.get('horario', '').strip()
        password = request.POST.get('password', '')
        password2 = request.POST.get('password2', '')

        # Campos obligatorios
        if not all([nombre, apellido, correo, telefono, direccion, rol, horario, password, password2]):
            messages.error(request, "Todos los campos son obligatorios.")
            return render(request, 'administrador/trabajadores/crear_trabajador.html')

        # Validar rol permitido (trabajadores)
        if rol not in ['mesero', 'cocinero', 'cajero']:
            messages.error(request, "Rol inválido.")
            return render(request, 'administrador/trabajadores/crear_trabajador.html')

        # Validar horario
        HORARIOS_VALIDOS = [
            'lv_manana', 'lv_tarde', 'lv_noche',
            'sab_manana', 'sab_tarde', 'dom_unico'
        ]
```

Figura 5.41. Views: CRUD de la tabla trabajadores.

5.3.5.1.4 Pruebas

Tabla 5.83. Caso de prueba: Crear usuario correctamente

Caso de Prueba – HU01 Crear usuario	
Código del Caso de Prueba	CP-01
Historia de Usuario	HU01 – Crear usuarios

Número de Iteración	Iteración 1
Nombre del Caso de Prueba	Crear usuario correctamente
Objetivo	Validar que el administrador pueda crear usuarios en el sistema
Usuario	Administrador
Precondiciones	El administrador debe haber iniciado sesión
Datos de Prueba	Nombre: Kataleya Apellido: Morales Correo: Kata@gmail.com Teléfono: 0992964222 Dirección: Eloy alfaro Rol: Mesero Horario de trabajo: Mañana (06:00 – 14:00) Contraseña: 123456 Confirmar contraseña: 123456
Pasos a Ejecutar	1. Ingresar al sistema como administrador. 2. Ir a módulo Usuarios. 3. Seleccionar “Crear usuario”. 4. Ingresar los datos. 5. Guardar.
Resultado Esperado	El usuario se registra correctamente y aparece en la lista
Resultado Obtenido	Registro correcto del usuario en la base de datos

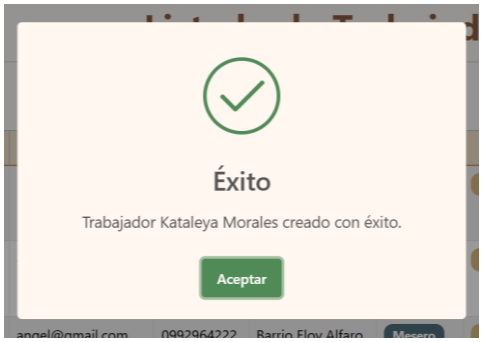
Evidencia	 Figura 5.42. Formulario de registro de un trabajador.  Figura 5.43. SweetAlert de registro exitoso del trabajador.

Tabla 5.84. Caso de prueba: Actualizar datos de un usuario.

Caso de Prueba – HU02 Actualizar usuarios	
Código del Caso de Prueba	CP-02
Historia de Usuario	HU02 – Actualizar usuarios
Número de Iteración	Iteración 1
Nombre del Caso de Prueba	Actualizar datos de un usuario

Objetivo	Validar que el administrador pueda modificar la información de los usuarios
Usuario	Administrador
Precondiciones	El administrador debe haber iniciado sesión y debe existir al menos un usuario registrado
Datos de Prueba	Nombre: Kataleya Apellido: Morales Correo: Kata@gmail.com Teléfono: 0992964222 Dirección: San Felipe Rol: Cajero Horario de trabajo: Mañana (06:00 – 14:00)
Pasos a Ejecutar	1. Ingresar al sistema como administrador. 2. Ir al módulo Usuarios. 3. Buscar el usuario a modificar. 4. Seleccionar opción “Editar”. 5. Cambiar los datos. 6. Guardar cambios.
Resultado Esperado	El sistema actualiza correctamente los datos del usuario y muestra la información nueva en la lista
Resultado Obtenido	Actualización correcta de los datos del usuario en el sistema

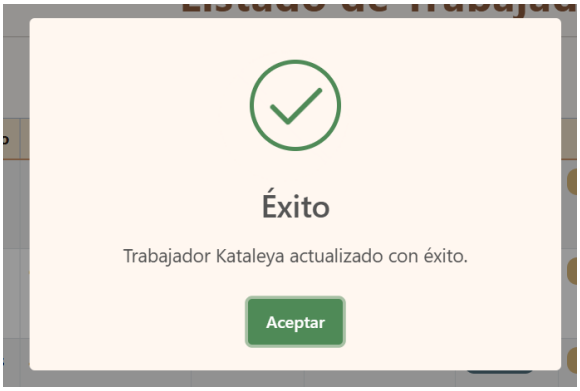
Evidencia	 Figura 5.44. Formulario de editar la información de un trabajador.  Figura 5.45. SweetAlert de actualización exitosa del trabajador.

Tabla 5.85. Caso de prueba: Actualizar datos de un usuario.

Caso de Prueba – HU03 Visualizar usuarios	
Código del Caso de Prueba	CP-03
Historia de Usuario	HU03 – Visualizar usuarios
Número de Iteración	Iteración 1

Nombre del Caso de Prueba	Visualizar listado de usuarios
Objetivo	Validar que el administrador pueda ver el listado completo de usuarios
Usuario	Administrador
Precondiciones	El administrador debe haber iniciado sesión y deben existir usuarios registrados
Datos de Prueba	Nombre: Kataleya Apellido: Morales Correo: Kata@gmail.com Teléfono: 0992964222 Dirección: San Felipe Rol: Cajero Horario de trabajo: Mañana (06:00 – 14:00) Estado: Activo
Pasos a Ejecutar	1. Ingresar al sistema como administrador. 2. Ir al módulo Usuarios. 3. Seleccionar opción “Listar usuarios”.
Resultado Esperado	El sistema muestra el listado de usuarios con sus datos principales (nombre, usuario, rol, estado)
Resultado Obtenido	Visualización de la información de los trabajadores registrados.

Evidencia	 Figura 5.46. Listado de los trabajadores.

Tabla 5.86. Caso de prueba: Desactivar usuario activo.

Caso de Prueba – HU04 Desactivar usuarios	
Código del Caso de Prueba	CP-04
Historia de Usuario	HU04 – Desactivar usuarios
Número de Iteración	Iteración 1
Nombre del Caso de Prueba	Desactivar usuario activo
Objetivo	Validar que el administrador pueda desactivar usuarios del sistema
Usuario	Administrador
Precondiciones	El administrador debe haber iniciado sesión y debe existir al menos un usuario activo
Datos de Prueba	Nombre: Kataleya Apellido: Morales Estado: Activo

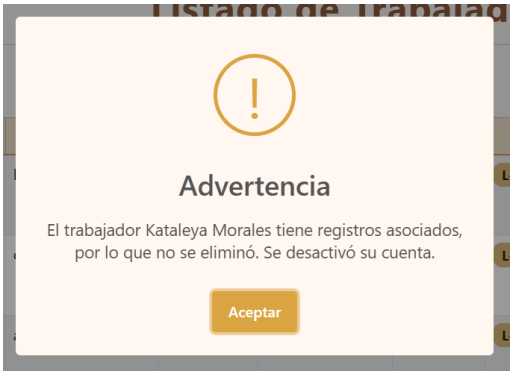
Pasos a Ejecutar	1. Ingresar al sistema como administrador. 2. Ir al módulo Usuarios. 3. Seleccionar opción “Listar usuarios”.
Resultado Esperado	El sistema muestra el listado de usuarios con sus datos principales (nombre, usuario, rol, estado)
Resultado Obtenido	Desactivar correctamente un trabajador en caso de su renuncia.
Evidencia	 Figura 5.47. SweetAlert para desactivar un trabajador.  Figura 5.48. SweetAlert que confirma la desactivación.

Tabla 5.87. Caso de prueba: Iniciar sesión con credenciales válidas.

Caso de Prueba – HU31 Iniciar sesión	
Código del Caso de Prueba	CP-05

Historia de Usuario	HU31 – Iniciar sesión
Número de Iteración	Iteración 1
Nombre del Caso de Prueba	Iniciar sesión con credenciales válidas
Objetivo	Verificar que el personal del restaurante pueda ingresar al sistema según su rol.
Usuario	Personal del restaurante
Precondiciones	El usuario debe estar registrado y activo en el sistema
Datos de Prueba	Correo: angel@gmail.com Contraseña: 123456
Pasos a Ejecutar	1. Abrir el sistema. 2. Ingresar usuario y contraseña. 3. Presionar el botón “Iniciar sesión”.
Resultado Esperado	El sistema permite el acceso y muestra las funciones según el rol del usuario
Resultado Obtenido	El sistema permitió al usuario iniciar sesión correctamente.

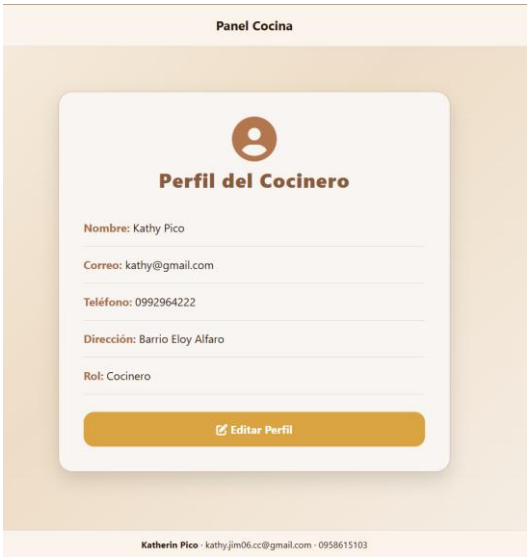
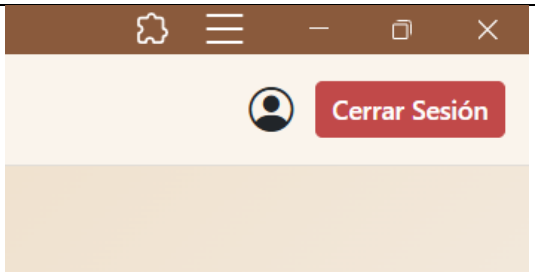
Evidencia	 Figura 5.49. Formulario de iniciar sesión.  Figura 5.50. Ingresar correctamente al sistema.

Tabla 5.88. Caso de prueba: Cerrar sesión correctamente.

Caso de Prueba – HU32 Cerrar sesión	
Código del Caso de Prueba	CP-06
Historia de Usuario	HU32 – Cerrar sesión

Número de Iteración	Iteración 1
Nombre del Caso de Prueba	Cerrar sesión correctamente
Objetivo	Verificar que el usuario pueda cerrar sesión de forma segura
Usuario	Personal del restaurante
Precondiciones	El usuario debe haber iniciado sesión
Datos de Prueba	Usuario activo: Kathy Pico
Pasos a Ejecutar	1. Ingresar al sistema con un usuario válido. 2. Presionar la opción “Cerrar sesión”.
Resultado Esperado	El sistema finaliza la sesión y redirige a la pantalla de inicio de sesión
Resultado Obtenido	El sistema permitió al usuario cerrar sesión exitosamente.
Evidencia	 Figura 5.51. Botón de cerrar sesión.

El desarrollo del sistema se llevó a cabo mediante varias iteraciones. La **iteración 2** estuvo orientada a la gestión de mesas del restaurante (Anexo C), seguida de la **iteración 3**, en la que se implementó la gestión del menú (Anexo D). Posteriormente, la **iteración 4** abordó la gestión de pedidos dentro del restaurante (Anexo E), mientras que la **iteración 5** se enfocó en la gestión de pedidos a domicilio (Anexo F). La **iteración 6** se centró en el módulo de cocina (Anexo G) y la **iteración 7** en la gestión de pagos y cobros (Anexo H). En la **iteración 8** se desarrollaron los reportes y el control administrativo (Anexo I), y en la **iteración 9** se incorporaron

funcionalidades relacionadas con el perfil de usuario, notificaciones y uso móvil (Anexo J). Finalmente, la **iteración 10** incluyó el módulo cliente para la realización de pedidos en línea (Anexo K) y la **iteración 11** correspondió a la validación de pagos en línea (Anexo L).

5.4 COSTO DEL SOFTWARE

Para estimar el costo de desarrollo del sistema se utilizó la técnica de estimación basada en puntos de historias de usuario (Story Points), la cual es ampliamente utilizada en metodologías ágiles como Extreme Programming (XP). Esta técnica permite evaluar el nivel de complejidad y esfuerzo requerido para implementar cada funcionalidad del sistema a partir de las historias de usuario definidas.

En la Tabla 5.41 se presentan todas las historias de usuario del sistema organizadas en orden secuencial junto con los puntos asignados a cada una. La suma de estos valores permite obtener una estimación global del esfuerzo requerido para el desarrollo completo de la aplicación.

Tabla 5.89. Estimación de puntos de historias de usuario.

Número	Historia de Usuario	Puntos
HU01	Crear usuarios	3
HU02	Actualizar usuarios	2
HU03	Visualizar usuarios	2
HU04	Desactivar usuarios	2
HU05	Registrar mesas	3
HU06	Registrar productos del menú	3
HU07	Visualizar reportes de ventas	3
HU08	Editar productos del menú	3
HU09	Desactivar productos del menú	2
HU10	Visualizar reportes por tipo de pedido	3
HU11	Visualizar reporte unificado de cobros	3
HU12	Editar perfil administrador	2
HU13	Editar perfil mesero	2

Número	Historia de Usuario	Puntos
HU14	Crear pedido para mesa	3
HU15	Agregar productos al pedido	3
HU16	Editar pedido	3
HU17	Eliminar pedido	2
HU18	Ver estado del pedido	2
HU19	Enviar pedido a cocina	2
HU20	Visualizar pedidos pendientes	2
HU21	Marcar pedido en preparación	2
HU22	Marcar pedido como listo	2
HU23	Avisar producto agotado	3
HU24	Editar perfil cocinero	3
HU25	Editar perfil cajero	3
HU26	Registrar pedido a domicilio	3
HU27	Registrar pago del pedido del restaurante	3
HU28	Ver total del pedido	2
HU29	Aplicar recargo por servicio	2
HU30	Generar comprobante de pago	3
HU31	Iniciar sesión	1
HU32	Cerrar sesión	1
HU33	Recibir notificaciones	2
HU34	Uso en dispositivos móviles	3
HU35	Visualizar pedidos a domicilio	2
HU36	Agregar productos a pedido a domicilio	3
HU37	Editar pedidos a domicilio	3

Número	Historia de Usuario	Puntos
HU38	Eliminar pedidos a domicilio	2
HU39	Enviar pedidos a domicilio a cocina	2
HU40	Visualizar estado de pedidos a domicilio	2
HU41	Registrar pago de pedidos a domicilio	2
HU42	Visualizar reporte unificado para cierre de caja	3
HU43	Visualizar mesas	2
HU44	Editar mesas	2
HU45	Eliminar mesas	2
HU46	Registro de cliente	3
HU47	Editar perfil cliente	2
HU48	Crear pedido	3
HU49	Agregar productos	3
HU50	Subir comprobante	3
HU51	Enviar pedido al cajero	2
HU52	Validar comprobante	3
HU53	Aprobar pedido	2
HU54	Rechazar pedido	2
Total de puntos		131

El total de puntos de historia será utilizado para calcular el tiempo estimado de desarrollo y el costo aproximado del software, considerando la relación entre puntos de historia, horas de desarrollo y costo por hora del desarrollador.

Para la estimación se establece la siguiente equivalencia:

1 punto de historia = 4 horas de desarrollo

Considerando que el sistema tiene un total de 131 puntos de historia, se realiza el siguiente cálculo:

$$Total\ puntos \times horas\ de\ desarrollo = total\ horas$$

$$131 \times 4 = 524\ horas$$

Por lo tanto, el tiempo estimado de desarrollo del sistema es de 524 horas.

Posteriormente, para estimar el costo de desarrollo se consideró el salario básico unificado que es USD. 482, el cual se dividió para 160 horas laborables al mes, dando un costo de USD. 3.01 por hora de trabajo.

$$Total\ horas \times costo\ por\ hora = costo\ del\ software$$

$$524 \times \$3.01 = \$1{,}577.24$$

En consecuencia, el costo total estimado del software es de USD. 1,577.24.

5.5 RESPUESTA A LA PREGUNTA DE INVESTIGACIÓN

La pregunta de investigación planteada fue: ¿Cuál es el nivel de usabilidad de la aplicación web progresiva desarrollada con Django para la gestión de pedidos y la comunicación interna en tiempo real en el restaurante “Productos Carlos Gerardo” de la ciudad del Tena?

Para responder esta pregunta se aplicó el test SUS (System Usability Scale) del Anexo M a 30 usuarios entre ellos los 12 trabajadores del restaurante y 18 clientes con el objetivo de evaluar la usabilidad del sistema desarrollado utilizando una escala del 1 al 5 como se muestra a continuación.

Tabla 5.90. Escala de SUS.

Valor	Significado
1	Totalmente en desacuerdo
2	En desacuerdo
3	Neutral
4	De acuerdo
5	Totalmente de acuerdo

Los resultados obtenidos en la pregunta 3 del cuestionario SUS, donde se evaluó si los usuarios consideraban que el sistema era fácil de usar, evidencian un alto nivel de aceptación en cuanto a la usabilidad de la aplicación. El 90% de los participantes seleccionó las opciones 4 y 5, lo que refleja que la mayoría de los usuarios percibe que el sistema es fácil de utilizar.

Estos resultados permiten responder a la pregunta de investigación indicando que la aplicación web progresiva desarrollada presenta un alto nivel de usabilidad, ya que los usuarios consideran que la interfaz es clara, intuitiva y comprensible para realizar las tareas relacionadas con la gestión de pedidos y la comunicación interna en tiempo real.

Por lo tanto, se puede afirmar que el sistema creado facilita la interacción de los usuarios con la plataforma, permitiendo que el equipo del restaurante “Productos Carlos Gerardo” utilice la aplicación de forma simple y efectiva para llevar a cabo sus labores cotidianas.

5.5.1 Análisis de resultados del Test SUS

5.5.1.1 Pregunta 1: Creo que me gustaría usar este sistema frecuentemente.

Tabla 5.91. Creo que me gustaría usar este sistema frecuentemente.

Respuesta	Frecuencia (fi)	Frecuencia relativa (fr)	Porcentaje (%)
1	1	0,033	3,3%
3	2	0,067	6,7%
4	6	0,200	20,0%
5	21	0,700	70,0%
Total	**30**	**1,00**	**100%**

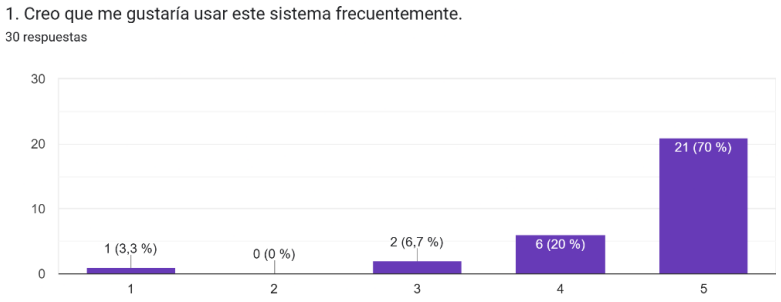


Figura 5.52. Creo que me gustaría usar este sistema frecuentemente.

Análisis:

El 90% de los participantes seleccionó los valores 4 y 5, lo que indica acuerdo o total acuerdo con la afirmación. Esto evidencia una alta disposición para el uso frecuente del sistema. Se observa una fuerte concentración en valores altos en preguntas positivas que refleja una percepción favorable de utilidad y aceptación del sistema.

5.5.1.2 Pregunta 2: Encontré el sistema innecesariamente complejo.

Tabla 5.92. Encontré el sistema innecesariamente complejo.

Respuesta	Frecuencia (fi)	Frecuencia relativa (fr)	Porcentaje (%)
1	20	0,667	66,7%
2	8	0,267	26,7%
3	1	0,033	3,3%
5	1	0,033	3,3%
Total	**30**	**1,00**	**100%**

Figura 5.53. Encontré el sistema innecesariamente complejo.

Análisis:

El 93,4% de los participantes seleccionó los valores 1 o 2, lo que claramente muestra un desacuerdo con la afirmación. Desde el punto de vista de la usabilidad, este resultado es muy alentador, ya que indica que la mayoría de los usuarios no ve el sistema como algo complicado. Esto sugiere que la interfaz es clara y fácil de entender.

5.5.1.3 Pregunta 3: Pensé que el sistema era fácil de usar.

Tabla 5.93. Pensé que el sistema era fácil de usar.

Respuesta	Frecuencia (fi)	Frecuencia relativa (fr)	Porcentaje (%)
2	1	0,033	3,3%
3	2	0,067	6,7%
4	9	0,300	30,0%
5	18	0,600	60,0%
Total	**30**	**1,00**	**100%**

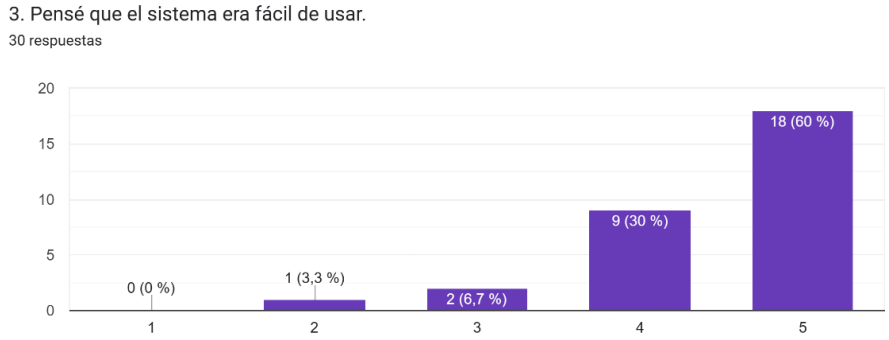


Figura 5.54. Pensé que el sistema era fácil de usar.

Análisis:

El 90% de los participantes respondió los valores 4 y 5, lo que indica que el sistema se considera fácil de usar. En cuanto a la usabilidad, este resultado refuerza la idea de que el diseño es intuitivo y se adapta bien tanto al personal como a los clientes.

5.5.1.4 Pregunta 4: Creó que necesitaría ayuda técnica para usar este sistema.

Tabla 5.94. Creó que necesitaría ayuda técnica para usar este sistema.

Respuesta	Frecuencia (fi)	Frecuencia relativa (fr)	Porcentaje (%)
1	21	0,700	70,0%
2	8	0,267	26,7%
4	1	0,033	3,3%
Total	**30**	**1,00**	**100%**

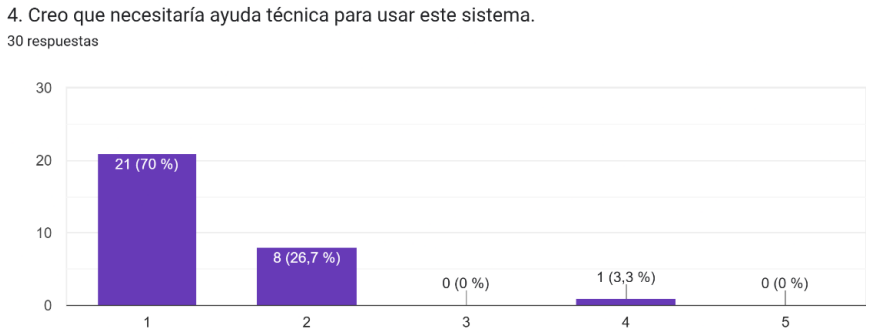


Figura 5.55. Creó que necesitaría ayuda técnica para usar este sistema.

Análisis:

El 96,7% de los participantes eligió los valores 1 o 2, lo que indica un claro desacuerdo. Este resultado es muy positivo en la escala SUS, ya que sugiere que los usuarios creen que el sistema se puede usar sin necesidad de asistencia técnica especializada, lo que significa que la curva de aprendizaje es bastante baja.

5.5.1.5 Pregunta 5: Las funciones del sistema están bien integradas.

Tabla 5.95. Las funciones del sistema están bien integradas..

Respuesta	Frecuencia (fi)	Frecuencia relativa (fr)	Porcentaje (%)
4	9	0,300	30,0%
5	21	0,700	70,0%
Total	**30**	**1,00**	**100%**

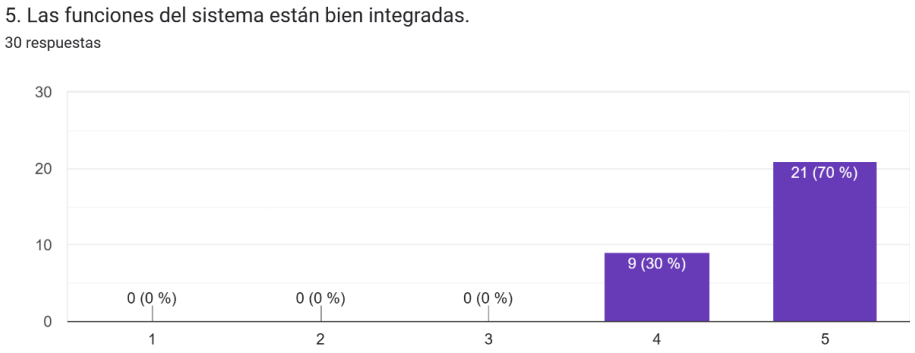


Figura 5.56. Las funciones del sistema están bien integradas.

Análisis:

El 100% de los participantes eligieron los valores de 4 o 5. Esto sugiere que los usuarios ven una buena coherencia y una integración adecuada entre las funciones del sistema, lo cual es fundamental para evaluar la usabilidad.

5.5.1.6 Pregunta 6: Encontré inconsistencias en el sistema.

Tabla 5.96. Encontré inconsistencias en el sistema.

Respuesta	Frecuencia (fi)	Frecuencia relativa (fr)	Porcentaje (%)
1	24	0,800	80,0%
2	5	0,167	16,7%
3	1	0,033	3,3%
Total	**30**	**1,00**	**100%**

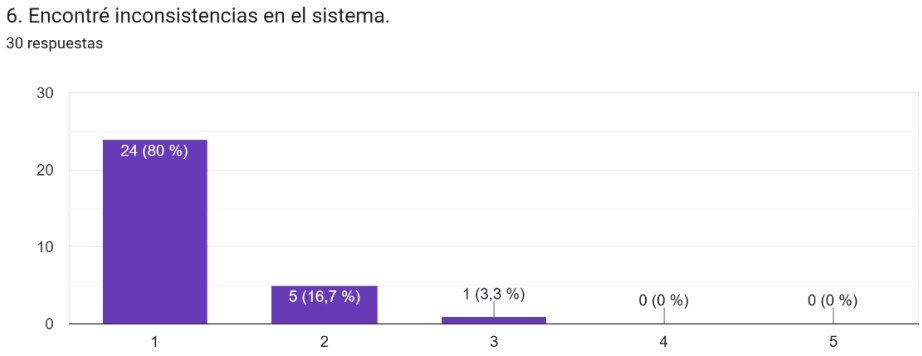


Figura 5.57. Encontré inconsistencias en el sistema.

Análisis:

El 96,7% respondió con los valores 1 o 2 esto demuestra que el sistema presenta estabilidad y consistencia en su funcionamiento. En la escala de SUS, este resultado refuerza la percepción de calidad y fiabilidad del sistema.

5.5.1.7 Pregunta 7: Creó que la mayoría de las personas aprenderían a usarlo rápidamente.

Tabla 5.97. Creó que la mayoría de las personas aprenderían a usarlo rápidamente.

Respuesta	Frecuencia (fi)	Frecuencia relativa (fr)	Porcentaje (%)
3	2	0,067	6,7%
4	9	0,300	30,0%
5	19	0,633	63,3%
Total	**30**	**1,00**	**100%**

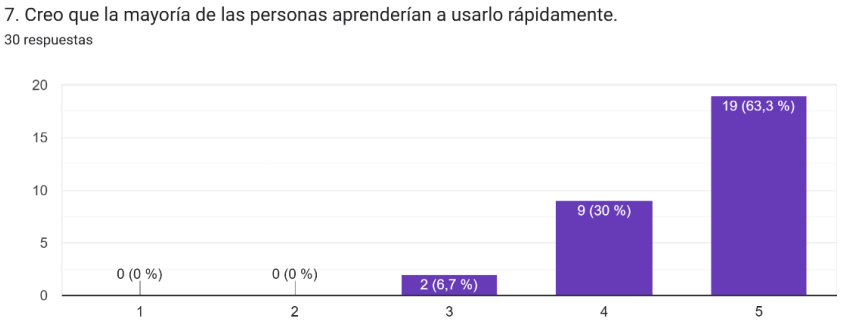


Figura 5.58. Creó que la mayoría de las personas aprenderían a usarlo rápidamente.

Análisis:

El 93,3% seleccionó los valores 4 o 5 esto indica que el sistema es fácil de aprender. Este resultado confirma una adecuada experiencia de usuario y simplicidad en la interacción.

5.5.1.8 Pregunta 8: El sistema me resultó complicado de usar.

Tabla 5.98. El sistema me resultó complicado de usar.

Respuesta	Frecuencia (fi)	Frecuencia relativa (fr)	Porcentaje (%)
1	23	0,767	76,7%
2	6	0,200	20,0%
3	1	0,033	3,3%
Total	**30**	**1,00**	**100%**

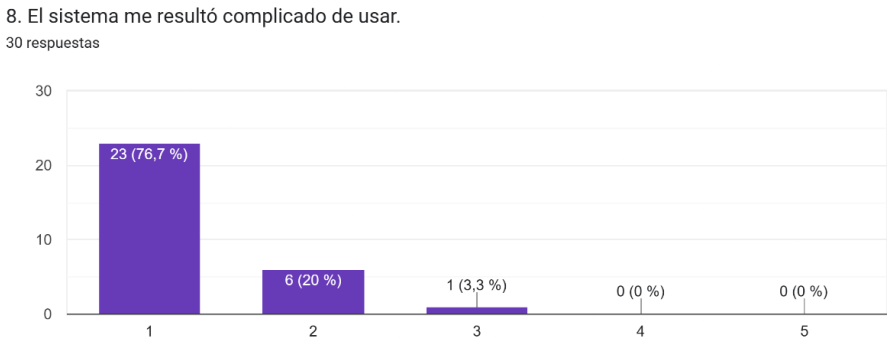


Figura 5.59. El sistema me resultó complicado de usar.

Análisis:

El 96,7% de los participantes seleccionó los valores 1 o 2; esto evidencia desacuerdo con la pregunta. Esto indica que los usuarios no consideran complicado el sistema.

5.5.1.9 Pregunta 9: Me sentí seguro al usar el sistema.

Tabla 5.99. Me sentí seguro al usar el sistema.

Respuesta	Frecuencia (fi)	Frecuencia relativa (fr)	Porcentaje (%)
3	2	0,067	6,7%
4	11	0,367	36,7%
5	17	0,567	56,7%
Total	**30**	**1,00**	**100%**

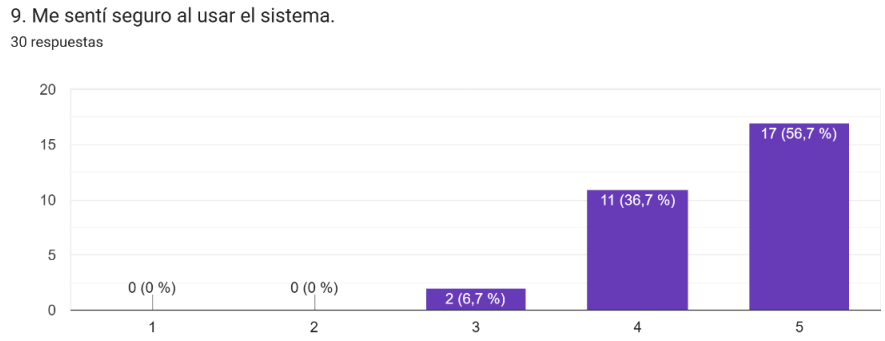


Figura 5.60. Me sentí seguro al usar el sistema.

Análisis:

El 93,4% de los participantes seleccionaron los valores 4 o 5, reflejando confianza en el uso del sistema.

5.5.1.10 Pregunta 10: Necesité aprender muchas cosas antes de poder usarlo.

Tabla 5.100. Necesité aprender muchas cosas antes de poder usarlo.

Respuesta	Frecuencia (fi)	Frecuencia relativa (fr)	Porcentaje (%)
1	22	0,733	73,3%
2	6	0,200	20,0%
3	2	0,067	6,7%
Total	**30**	**1,00**	**100%**

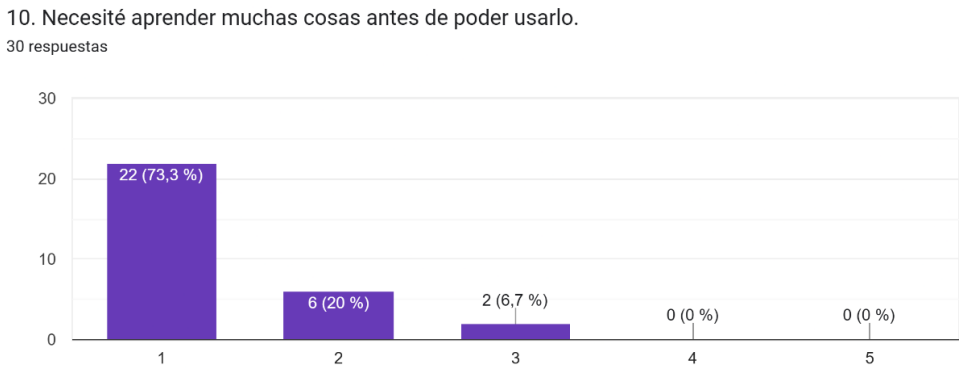


Figura 5.61. Necesité aprender muchas cosas antes de poder usarlo.

Análisis:

El 93,3% de los participantes respondieron los valores 1 o 2, lo que indica que el sistema no requiere aprendizaje complejo previo. Este resultado confirma que la curva de aprendizaje es baja y que la interfaz es intuitiva.

6. CONCLUSIONES Y RECOMENDACIONES

6.1 CONCLUSIONES

- El análisis de los fundamentos teóricos sobre la metodología XP y la programación web progresiva (PWA) realizada mediante la revisión bibliográfica permitió comprender adecuadamente la fundamentación teórica del proyecto, demostrando que la combinación de una metodología ágil con tecnologías web modernas es lo más adecuado para el desarrollo de propuestas tecnológicas en microempresas.
- La aplicación de la metodología XP permitió desarrollar el sistema mediante iteraciones cortas, facilitando la construcción progresiva de los diferentes módulos de la aplicación. Este enfoque permitió adaptar el desarrollo a las necesidades del restaurante y obtener un sistema funcional que mejora la gestión de pedidos y la comunicación entre las áreas del establecimiento.
- La implementación de la aplicación web acompañada de pruebas de rendimiento demostró que el sistema cumple con los requerimientos establecidos y demostrando una alta aceptación de la aplicación por parte del personal del establecimiento.

6.2 RECOMENDACIONES

- Implementar la aplicación web progresiva en el entorno real del restaurante durante un periodo prolongado, con el fin de evaluar su funcionamiento en las actividades diarias del establecimiento. Esto permitirá identificar posibles mejoras en el proceso de gestión de pedidos y optimizar la comunicación entre las diferentes áreas del restaurante.
- Realizar capacitaciones básicas al personal del restaurante sobre el uso de la aplicación, con el propósito de garantizar una correcta utilización del sistema y aprovechar adecuadamente sus funcionalidades en la gestión de pedidos y comunicación entre las áreas de trabajo.
- Considerar la incorporación de nuevas funcionalidades en futuras versiones del sistema, como la integración con plataformas de pago digital, generación de reportes más avanzados o notificaciones automáticas para los pedidos, con el propósito de mejorar la eficiencia operativa del restaurante y brindar una mejor experiencia a los usuarios

7. REFERENCIAS

[1] Fernando Luna, *PWA: Desarrolla Aplicaciones Web Multi Dispositivos - Vol.1: Implementa las bases de una PWA*, Claudio Peña., vol. 1. Ciudad Autónoma de Buenos Aires: RedUsers, 2021. Accedido: 7 de octubre de 2025. [En línea]. Disponible en: https://books.google.com.ec/books?id=eQg1EAAAQBAJ&printsec=frontcover&dq=isbn:9789878414140&hl=&cd=1&source=gbs_api&redir_esc=y#v=onepage&q&f=false

[2] Marycarmen Díaz y Antonio Collazo, «La Programación Extrema», 2013.

[3] K. B. Pozo-Benites, K. W. Guadalupe-Sánchez, E. E. Peñarreta-Barrera, y J. K. Meza-Salvatierra, «Transformación digital de las PYMES en América Latina: barreras, oportunidades y estrategias para la competitividad», *Multidisciplinary Latin American Journal (MLAJ)*, vol. 3, n.º 2, pp. 236-255, jun. 2025, doi: 10.62131/mlaj-v3-n2-015.

[4] «El 93,67% de empresas ecuatorianas son micro; su situación es compleja». Accedido: 7 de octubre de 2025. [En línea]. Disponible en: https://www.radiopichincha.com/el-9367-de-empresas-ecuatorianas-son-micro-su-situacion-es-compleja/

[5] Alanuca Añarumba Carlos Alexander, «APLICACIÓN MÓVIL PARA PEDIDOS DE COMIDA A DOMICILIO EN LA CIUDAD DE LATACUNGA.», UNIVERSIDAD TÉCNICA DE COTOPAXI, 2021.

[6] Karla Gonzáles, «Sistema web para la automatización del servicio en restaurantes», Tecnológico Nacional de México, 2022.

[7] Pezantes Carpio y Diana Marilu, «Aplicación web para la gestión del servicio al cliente en el restaurante innovatión food de la ciudad de Ambato», Universidad Regional Autónoma de Los Andes, Ambato, 2017. Accedido: 7 de octubre de 2025. [En línea]. Disponible en: http://dspace.uniandes.edu.ec/handle/123456789/7382

[8] J. Zofio Jimenez, *Aplicaciones web*. Macmillan Iberia, S A, 2013. Accedido: 10 de diciembre de 2025. [En línea]. Disponible en: https://elibro.net/es/ereader/utcotopaxi/43262

[9] Jose Luis Muñoz Tapia, «DESARROLLO E IMPLEMENTACIÓN DE UNA APLICACIÓN WEB PROGRESIVA (PWA)», Universitat Politècnica de Catalunya, Barcelona, 2020.

[10] Llamuca Quinaloa y Yasmin Vera, «Arquitectura de una PWA.», jun. 2021.

[11] Alex Bladimir; Gissella Alcivar; Carmen Moreira, «IMPLEMENTACIÓN DE ACCESIBILIDAD EN PROGRESSIVE WEB APPS (PWAs): DESAFÍOS, TÉCNICAS Y BENEFICIOS EN LA UNIVERSIDAD ESTATAL DE MANABÍ ECUADOR», *Revista Científica de Innovación Educativa y Sociedad Actual «ALCON»*, Manabí, 2023.

[12] «PWA o Aplicación Web Progresiva: qué son y por qué están transformando el desarrollo multiplataforma». Accedido: 11 de diciembre de 2026. [En línea]. Disponible en: https://universidadeuropea.com/blog/pwa-aplicacion-web-progresiva/#diferencias-entre-aplicaciones-web-aplicaciones-nativas-y-pwa

[13] Mozilla Developer Network, «Introducción a Django». Accedido: 17 de diciembre de 2025. [En línea]. Disponible en: https://developer.mozilla.org/es/docs/Learn_web_development/Extensions/Server-side/Django/Introduction#%C2%BFqu%C3%A9_es_django

[14] Chen Songtao, Ahmmed Shahed, Lal Karu, y Deming Chunhua, «Django Web Development Framework: Powering the Modern Web», *American Journal of Trade and Policy*, vol. 7, n.º 3, pp. 99-106, dic. 2020, doi: 10.18034/ajtp.v7i3.675.

[15] Read the Docs, «Channels Documentation». Accedido: 28 de diciembre de 2025. [En línea]. Disponible en: https://channels.readthedocs.io/en/latest/

[16] IBM, «IBM WebSphere Application Server — WebSocket applications». Accedido: 29 de diciembre de 2025. [En línea]. Disponible en: https://www.ibm.com/docs/es/was/9.0.5?topic=applications-websocket

[17] C. L. Vidal-Silva, A. Sánchez-Ortiz, J. Serrano, y J. M. Rubio, «Experiencia académica en desarrollo rápido de sistemas de información web con Python y Django», *Formación universitaria*, vol. 14, n.º 5, pp. 85-94, oct. 2021, doi: 10.4067/S0718-50062021000500085.

[18] Python Software Foundation, «Python — The Python Programming Language». Accedido: 22 de diciembre de 2025. [En línea]. Disponible en: Python Software Foundation

[19] Ainoa. Celaya Luna, *Creación de páginas web : HTML 5*. ICB Editores, 2015.

[20] Cursos GIS, «¿Cómo integramos los lenguajes HTML, CSS y JavaScript?» Accedido: 27 de diciembre de 2025. [En línea]. Disponible en: https://www.cursosgis.com/como-integramos-los-lenguajes-html-css-y-javascript/#

[21] Miguel. Hernández Bejarano y L. Eduardo. Baquero Rey, *Fundamentos de programación web. 1*. Editorial Universidad ECCI, 2023.

[22] Picuino, «Índice de CSS». Accedido: 27 de diciembre de 2025. [En línea]. Disponible en: https://www.picuino.com/es/css-index.html

[23] Alfonso. Urquía Moraleda, *Lenguajes de programación*. UNED - Universidad Nacional de Educación a Distancia, 2021.

[24] People Like Us, «What is JavaScript and why is it required for people like us». Accedido: 28 de diciembre de 2025. [En línea]. Disponible en: https://peoplelikeus.world/es/blog/what-is-javascript-and-why-is-it-required-for-people-like-us

[25] R. E. . Beasley, *Essential ASP.NET Web Forms Development : Full Stack Programming with C#, SQL, Ajax, and JavaScript*. Apress, 2020.

[26] Raiola Networks, «Bootstrap: ¿Qué es?» Accedido: 28 de diciembre de 2025. [En línea]. Disponible en: https://raiolanetworks.com/blog/bootstrap/#que-es-

[27] The Bootstrap Authors, «Bootstrap». Accedido: 28 de diciembre de 2025. [En línea]. Disponible en: https://getbootstrap.com/

[28] Arsys, «¿Qué es Visual Studio Code y cuáles son sus ventajas?» Accedido: 29 de diciembre de 2025. [En línea]. Disponible en: https://www.arsys.es/blog/que-es-visual-studio-code-y-cuales-son-sus-ventajas

[29] Microsoft, «Visual Studio Code icon», https://code.visualstudio.com/brand. Accedido: 29 de diciembre de 2025. [En línea]. Disponible en: https://commons.wikimedia.org/wiki/File:Visual_Studio_Code_1.35_icon.svg

[30] INESEM, «Los gestores de bases de datos más usados». Accedido: 29 de diciembre de 2025. [En línea]. Disponible en: https://www.inesem.es/revistadigital/informatica-y-tics/los-gestores-de-bases-de-datos-mas-usados

[31] Sarah Laoyan, «¿Qué es la Metodología Agile y cómo revoluciona la gestión de proyectos?» Accedido: 29 de diciembre de 2025. [En línea]. Disponible en: https://asana.com/es/resources/agile-methodology

[32] Alicia Raeburn, «Extreme Programming (XP)». Accedido: 29 de diciembre de 2025. [En línea]. Disponible en: https://asana.com/es/resources/extreme-programming-xp

[33] C. G. A. Díaz Labrador Marycarmen, «La Programación Extrema», nov. 2013.

[34] P. Letelie y C. Penadé, «Métodologías ágiles para el desarrollo de software: eXtreme Programming (XP)», vol. 05, 2002.

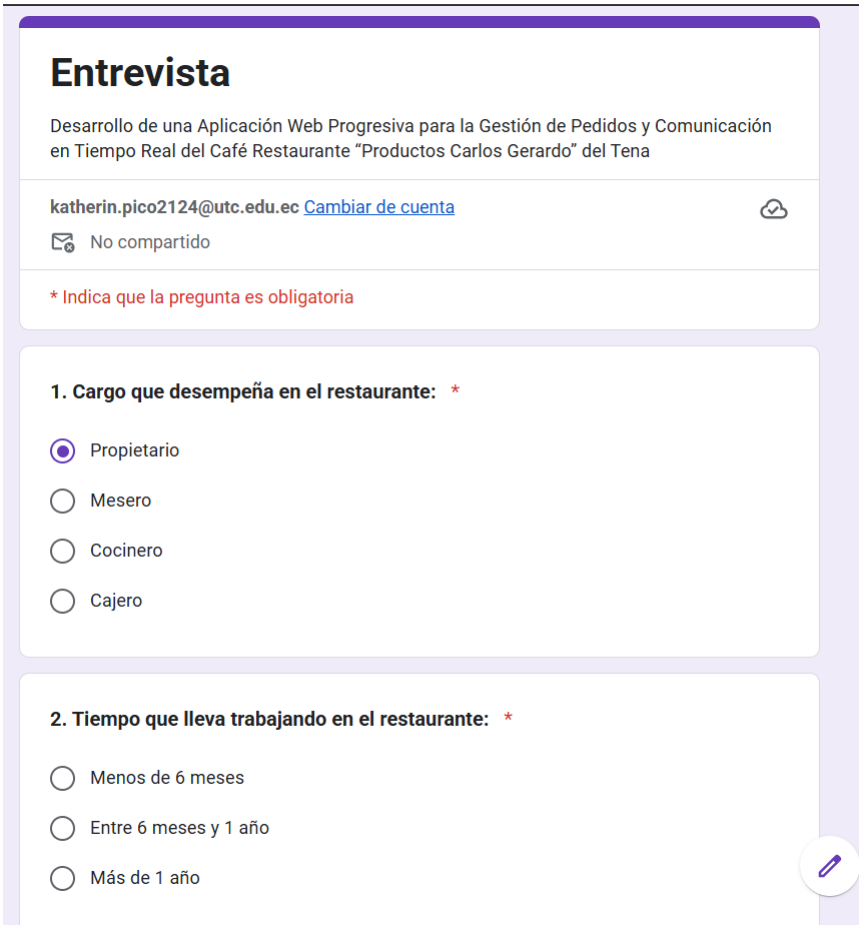
[35] D. Martín-Martín, J. M. García, y I. Romero, «DETERMINANTS OF DIGITAL TRANSFORMATION IN THE RESTAURANT INDUSTRY», *Amfiteatru Economic*, vol. 24, n.º 60, pp. 430-446, 2022, doi: 10.24818/EA/2022/60/430.

[36] A. C. Katagri y K. B. Balavalad, «Smart Restaurant Ordering: A Digital Transformation to Enhance Dining Experiences», 2025.

[37] Gastrouni, «Guía rápida para gestionar tu restaurante como un profesional», 2018. Accedido: 17 de diciembre de 2026. [En línea]. Disponible en: https://www.gastrouni.com/wp-content/uploads/2018/08/ebook-gastrouni-guia-gestion-restaurante-como-profesional.pdf

8. ANEXOS

ANEXO A: FORMULARIO DE LA ENTREVISTA SEMIESTRUCTURADA



Entrevista

Desarrollo de una Aplicación Web Progresiva para la Gestión de Pedidos y Comunicación en Tiempo Real del Café Restaurante "Productos Carlos Gerardo" del Tena

katherin.pico2124@utc.edu.ec Cambiar de cuenta

No compartido

* Indica que la pregunta es obligatoria

1. Cargo que desempeña en el restaurante: *

- Propietario
- Mesero
- Cocinero
- Cajero

2. Tiempo que lleva trabajando en el restaurante: *

- Menos de 6 meses
- Entre 6 meses y 1 año
- Más de 1 año

3. ¿Cómo se realiza actualmente la toma y gestión de pedidos en el restaurante? *

Tu respuesta

4. ¿Qué problemas o dificultades se presentan con el método actual de trabajo? *

Tu respuesta

5. ¿En qué etapas del proceso considera que se producen más errores o retrasos? *

Tu respuesta

6. ¿Cómo es la comunicación entre meseros, cocina y caja durante horas de mayor demanda? *

Tu respuesta

7. ¿Qué consecuencias genera una comunicación lenta o poco clara entre las áreas? *

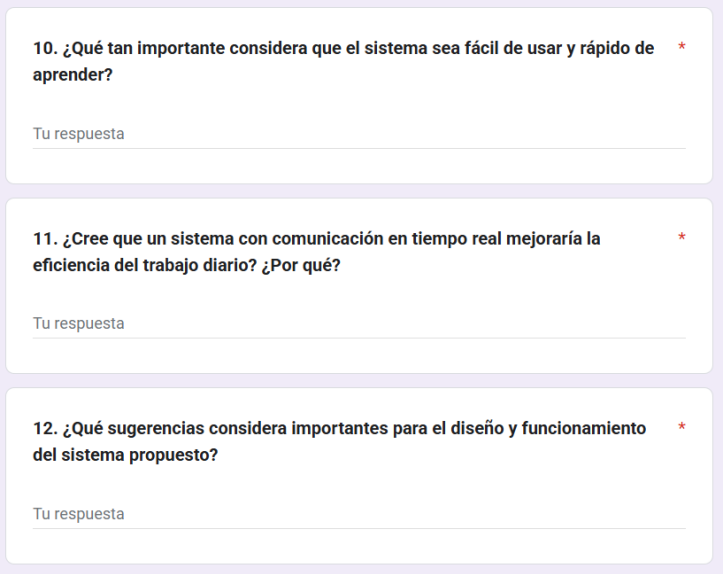
Tu respuesta

8. ¿Qué tareas considera que podrían automatizarse mediante un sistema web? *

Tu respuesta

9. ¿Qué información considera indispensable que el sistema muestre para realizar su trabajo? *

Tu respuesta



10. ¿Qué tan importante considera que el sistema sea fácil de usar y rápido de aprender? *

Tu respuesta

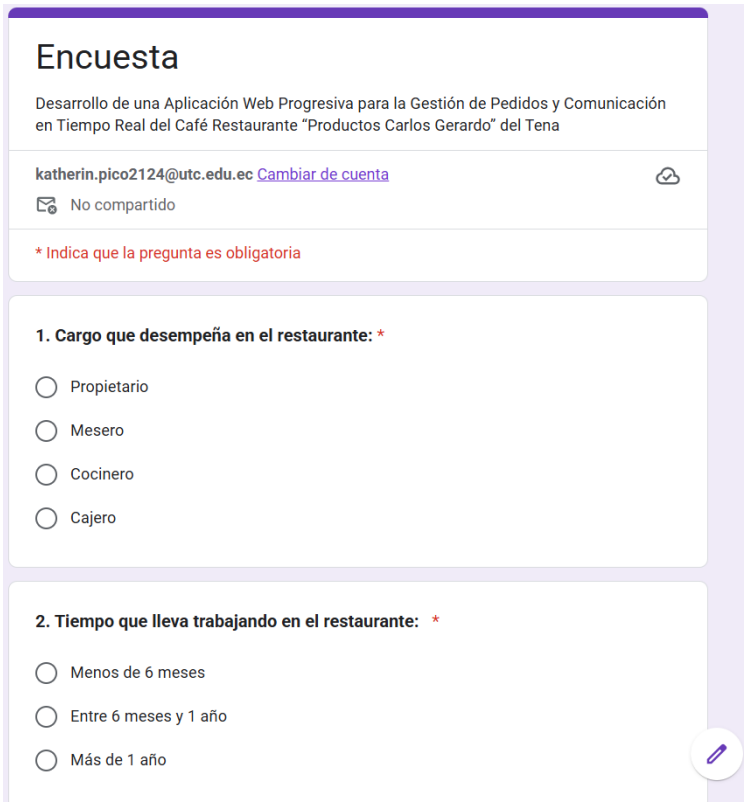
11. ¿Cree que un sistema con comunicación en tiempo real mejoraría la eficiencia del trabajo diario? ¿Por qué? *

Tu respuesta

12. ¿Qué sugerencias considera importantes para el diseño y funcionamiento del sistema propuesto? *

Tu respuesta

ANEXO B: FORMULARIO DE LA ENCUESTA ESTRUCTURADA



Encuesta

Desarrollo de una Aplicación Web Progresiva para la Gestión de Pedidos y Comunicación en Tiempo Real del Café Restaurante "Productos Carlos Gerardo" del Tena

katherin.pico2124@utc.edu.ec Cambiar de cuenta

No compartido

* Indica que la pregunta es obligatoria

1. Cargo que desempeña en el restaurante: *

- ◯ Propietario
- ◯ Mesero
- ◯ Cocinero
- ◯ Cajero

2. Tiempo que lleva trabajando en el restaurante: *

- ◯ Menos de 6 meses
- ◯ Entre 6 meses y 1 año
- ◯ Más de 1 año

3. ¿Considera necesario implementar un sistema web para la gestión de pedidos * en el restaurante?

- ◯ Sí
- ◯ No

4. ¿Cree que un sistema web reduciría los errores en la toma de pedidos? *

- ◯ Sí
- ◯ No

5. ¿Un sistema con comunicación en tiempo real mejoraría la coordinación entre * las áreas?

- ◯ Sí
- ◯ No

6. ¿Considera importante que el sistema sea fácil de usar? *

- ◯ Sí
- ◯ No

7. ¿Cree que aprender a usar el sistema sería sencillo para el personal? *

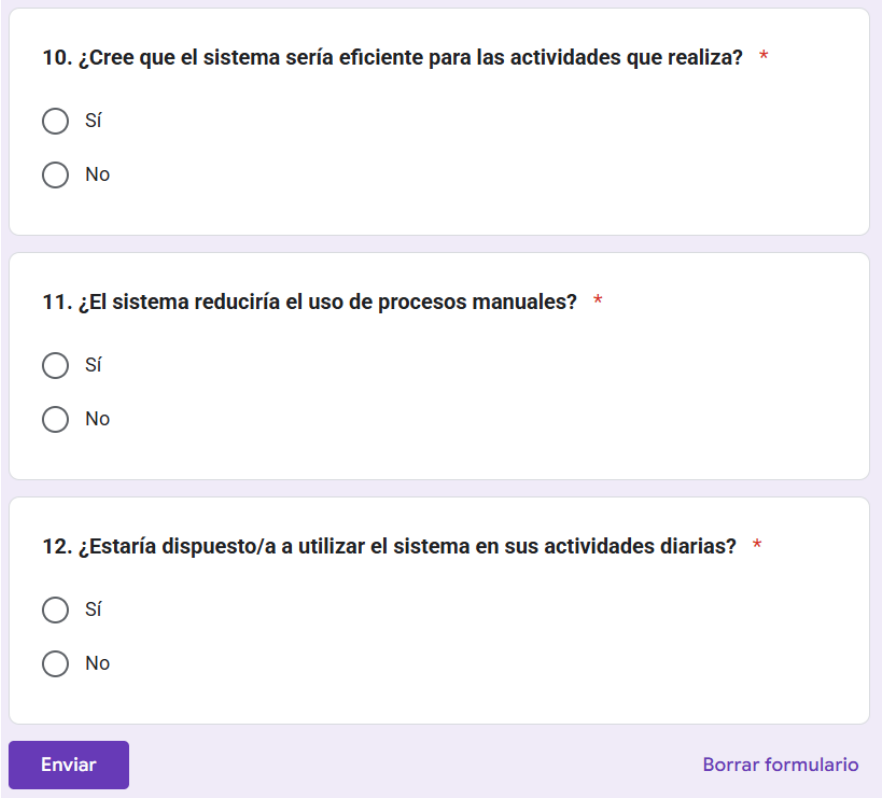
- ◯ Sí
- ◯ No

8. ¿El uso del sistema permitiría atender a los clientes de forma más rápida? *

- ◯ Sí
- ◯ No

9 ¿Considera que el sistema mejoraría la organización del trabajo diario? *

- ◯ Sí
- ◯ No



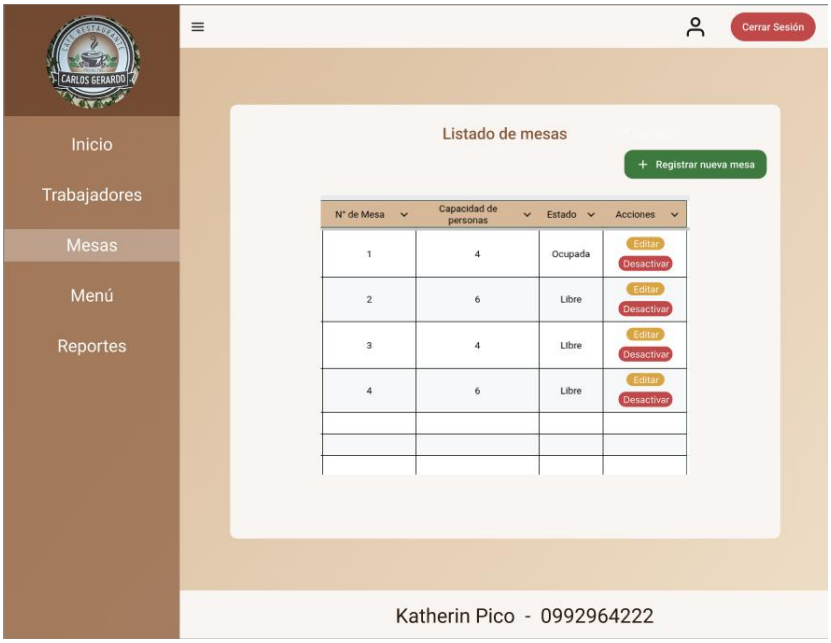
ANEXO C: ITERACIÓN 2 – GESTIÓN DE MESAS DEL RESTAURANTE

PLANIFICACIÓN

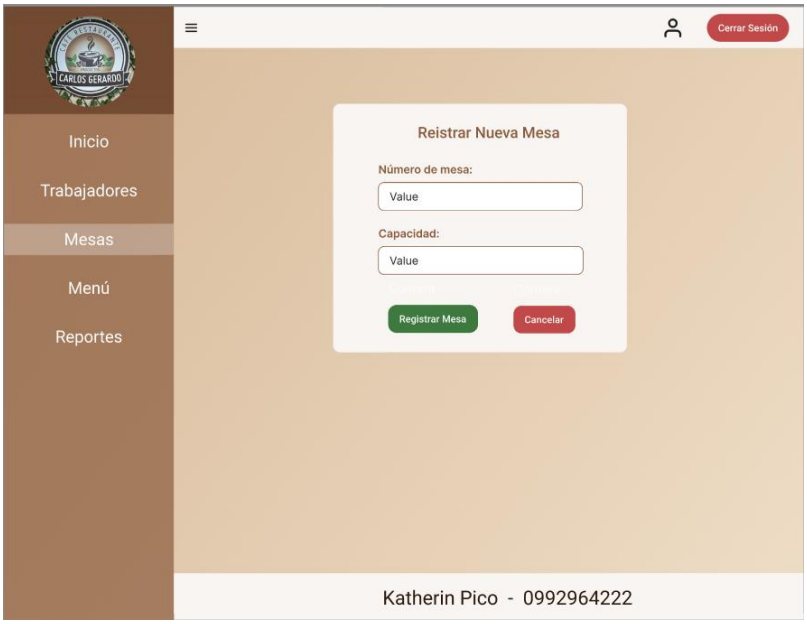
Número	Historia de Usuario
HU05	Registrar mesas
HU43	Visualizar mesas
HU44	Editar mesas
HU45	Eliminar mesas

DISEÑO

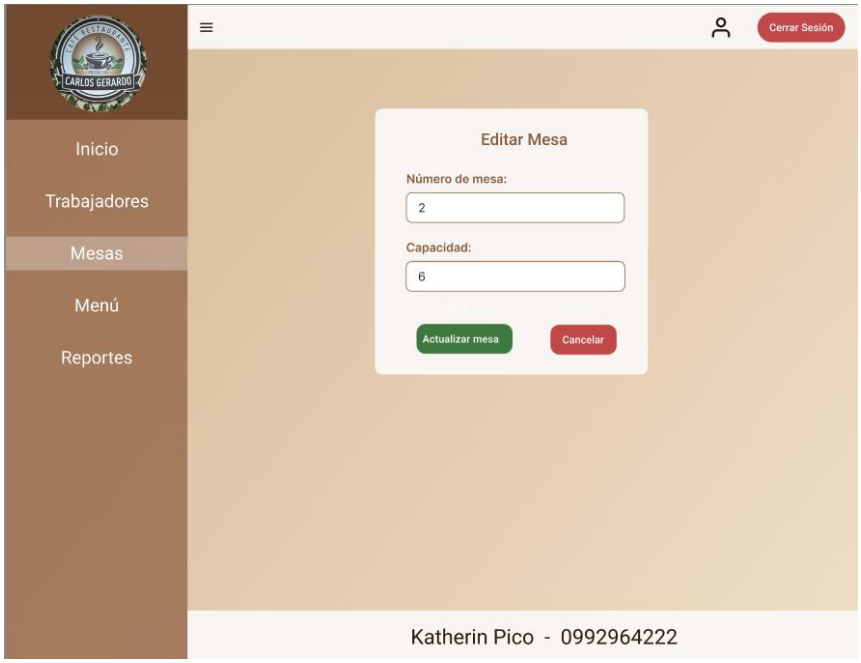
Se observa el listado de las mesas del restaurante las cuales ya están registradas en el sistema.



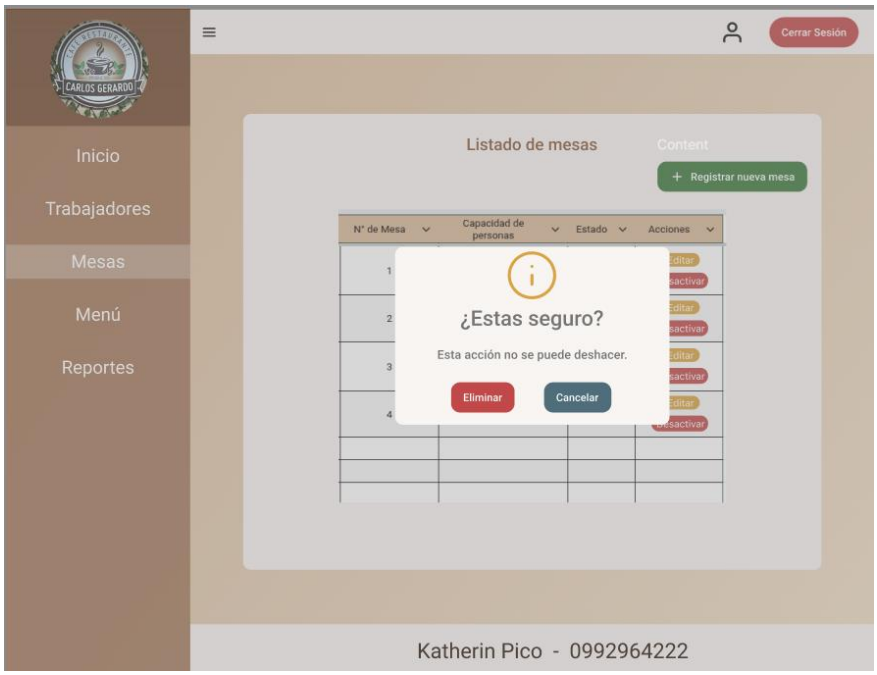
Se observa el formulario de registro de mesas del restaurante, en donde el administrador ingresa la capacidad de personas que permite dicha mesa registrada.



Se observa el formulario de editar la información registrada de las mesas de restaurante.



Se observa el mensaje de confirmación para eliminar una mesa registrada en caso de que su estado esté en libre.



CODIFICACIÓN

Se observa el código fuente del template de la gestión de mesas realizada por el administrador.

```
<div class="main-content-wrapper">
    <div class="container mt-5">
        <div class="content-card">
            </div>

            <div class="table-responsive">
                <table class="table table-bordered table-striped table-hover" id="tbl_mesas">
                    <thead class="table-primary tex" >
                        <tr>
                            <th>N° de Mesa</th>
                            <th>Capacidad de personas</th>
                            <th>Estado</th>
                            <th>Acciones</th>
                        </tr>
                    </thead>
                    <tbody>
                        {% for mesa in mesas %}
                        <tr>
                            <td>{{ mesa.numero }}</td>
                            <td>{{ mesa.capacidad }}</td>
                            <td>{{ mesa.estado }}</td>
                            <td>
                                <a href="{% url 'editar_mesa' mesa.id %}" class="btn btn-warning btn-sm me-1">
                                    <i class="fas fa-edit"></i> Editar
                                </a>

                                <form method="post" action="{% url 'eliminar_mesa' mesa.id %}" style="display:inline;">
                                    {% csrf_token %}
                                    <button type="submit" class="btn btn-danger btn-sm"
                                        onclick="return confirmarEliminacion(event);">
                                        <i class="fas fa-trash"></i> Eliminar
                                    </button>
                                </form>

                            </td>
                        </tr>
                        {% empty %}
                        <tr>
                            <td colspan="4" class="text-center text-muted py-4">
                                No hay mesas registradas.
                            </td>
                        </tr>
                        {% endfor %}
                    </tbody>
                </table>
```

Se observa el código fuente del models en donde se está creando la tabla de mesas para el registro en la base de datos.

```python
# ------------------------------
# Mesa
# ------------------------------
class Mesa(models.Model):
    numero = models.PositiveIntegerField(unique=True)
    capacidad = models.PositiveIntegerField(default=4)
    estado = models.CharField(max_length=20, choices=[
        ('libre', 'Libre'),
        ('ocupada', 'Ocupada'),
    ], default='libre')

    def __str__(self):
        return f"Mesa {self.numero} ({self.estado})"
```

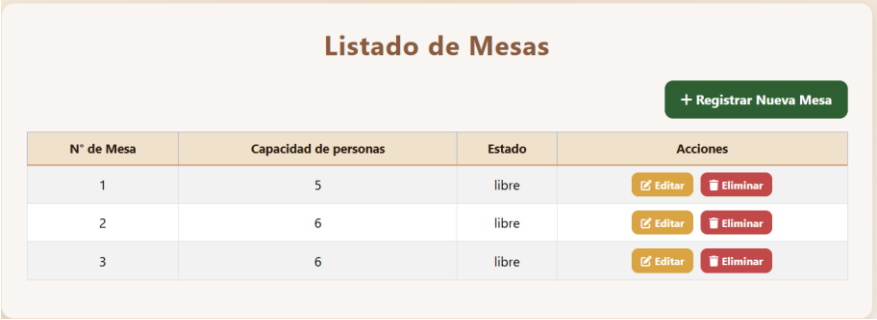
Se observa el código del views en donde se programa la lógica del backend para el CRUD de las mesas registradas por el administrador.

```python
# Listar mesas
@login_required(login_url='login')
@rol_requerido('admin')
def listar_mesas(request):
    mesas = Mesa.objects.all().order_by('numero')
    return render(request, 'administrador/mesas/listar_mesas.html', {'mesas': mesas})

#crear una mesa
@login_required(login_url='login')
@rol_requerido('admin')
def registrar_mesa(request):
    # Obtener el último número de mesa
    ultima_mesa = Mesa.objects.order_by('-numero').first()
    siguiente_numero = ultima_mesa.numero + 1 if ultima_mesa else 1

    if request.method == 'POST':
        numero = request.POST.get('numero')
        capacidad = request.POST.get('capacidad')

        # Validar campos vacíos
        if not numero or not capacidad:
            messages.error(request, "Todos los campos son obligatorios.")
            return render(request, 'administrador/mesas/registrar_mesa.html', {
                'siguiente_numero': siguiente_numero
            })

        # Validar que sean números enteros
        if not numero.isdigit() or not capacidad.isdigit():
            messages.error(request, "El número y la capacidad deben ser valores numéricos.")
            return render(request, 'administrador/mesas/registrar_mesa.html', {
                'siguiente_numero': siguiente_numero
            })

        # Convertir a int para validaciones
        numero = int(numero)
        capacidad = int(capacidad)

        # Validar que sean positivos
        if numero < 1 or capacidad < 1:
            messages.error(request, "Número y capacidad deben ser mayores o iguales a 1.")
            return render(request, 'administrador/mesas/registrar_mesa.html', {
                'siguiente_numero': siguiente_numero
            })

```

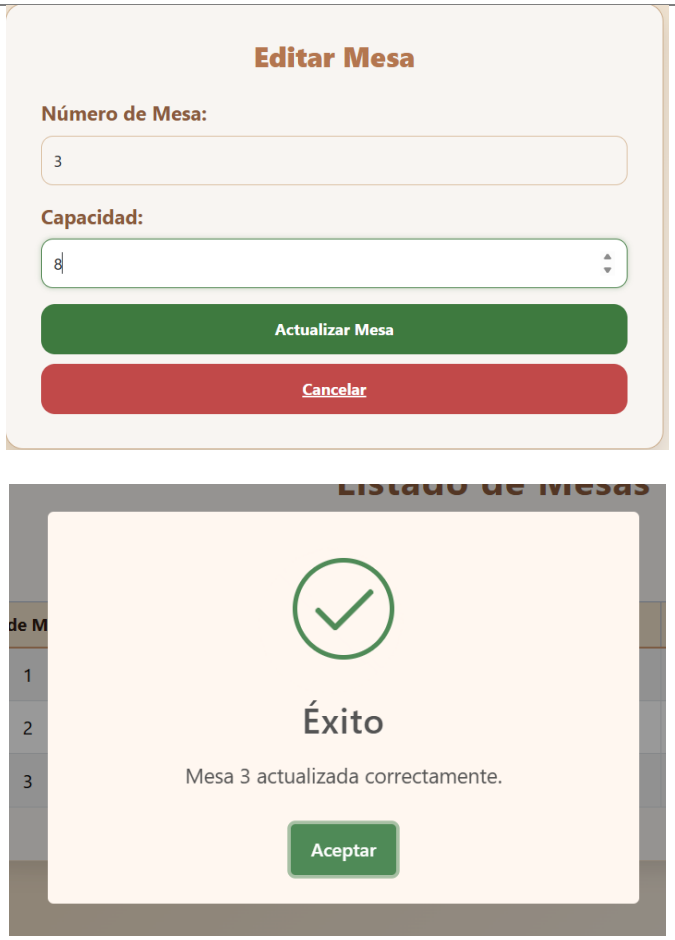
PRUEBAS

Caso de Prueba – HU05 Registrar mesas	
Código del Caso de Prueba	CP-07
Historia de Usuario	HU05 – Registrar mesas
Nombre del Caso de Prueba	Registrar nueva mesa

Objetivo	Validar que el administrador pueda registrar mesas en el sistema
Usuario	Administrador
Precondiciones	El administrador debe haber iniciado sesión
Datos de Prueba	Número de mesa: 3 Capacidad: 6 personas
Pasos a Ejecutar	1. Ingresar al sistema como administrador. 2. Ir al módulo Mesas. 3. Seleccionar “Registrar mesa”. 4. Ingresar los datos de la mesa. 5. Guardar.
Resultado Esperado	La mesa se registra correctamente y aparece en el listado de mesas
Resultado Obtenido	El sistema registro correctamente la mesa.
Evidencia	 

Caso de Prueba – HU43 Visualizar mesas	
Código del Caso de Prueba	CP-08
Historia de Usuario	HU43 – Visualizar mesas
Nombre del Caso de Prueba	Visualizar listado de mesas
Objetivo	Verificar que el administrador pueda ver todas las mesas registradas
Usuario	Administrador
Precondiciones	El administrador debe haber iniciado sesión y deben existir mesas registradas
Datos de Prueba	Mesas existentes: Mesa 1 (4 personas), Mesa 2 (2 personas), Mesa 3 (6 personas)
Pasos a Ejecutar	1. Ingresar al sistema como administrador. 2. Ir al módulo Mesas. 3. Seleccionar opción “Listar mesas”.
Resultado Esperado	El sistema muestra el listado de mesas con número, capacidad y estado
Resultado Obtenido	El sistema muestra de forma correcta el listado de las mesas registradas en el sistema.
Evidencia	

Caso de Prueba – HU44 Editar mesas	
Código del Caso de Prueba	CP-09
Historia de Usuario	HU44 – Editar mesas
Nombre del Caso de Prueba	Editar datos de una mesa
Objetivo	Validar que el administrador pueda modificar la información de las mesas
Usuario	Administrador
Precondiciones	El administrador debe haber iniciado sesión y debe existir al menos una mesa registrada
Datos de Prueba	Mesa a editar: Mesa 3 Nueva capacidad: 8 personas
Pasos a Ejecutar	1. Ingresar al sistema como administrador. 2. Ir al módulo Mesas. 3. Buscar la mesa a modificar. 4. Seleccionar opción “Editar”. 5. Cambiar los datos. 6. Guardar cambios.
Resultado Esperado	El sistema actualiza correctamente los datos de la mesa y muestra la información nueva
Resultado Obtenido	El sistema edita la capacidad de personas de la mesa.

Evidencia	

Caso de Prueba – HU45 Eliminar mesas	
Código del Caso de Prueba	CP-10
Historia de Usuario	HU45 – Eliminar mesas
Nombre del Caso de Prueba	Eliminar mesa no utilizada
Objetivo	Validar que el administrador pueda eliminar mesas que no estén en uso
Usuario	Administrador

Precondiciones	El administrador debe haber iniciado sesión y debe existir al menos una mesa registrada que no esté asignada a un pedido
Datos de Prueba	Mesa a eliminar: Mesa 3 (estado: disponible, sin pedidos activos)
Pasos a Ejecutar	1. Ingresar al sistema como administrador. 2. Ir al módulo Mesas. 3. Buscar la mesa que no esté en uso. 4. Seleccionar opción “Eliminar”. 5. Confirmar la acción
Resultado Esperado	La mesa se elimina correctamente y ya no aparece en el listado
Resultado Obtenido	El sistema elimina correctamente una mesa registrada.
Evidencia	 

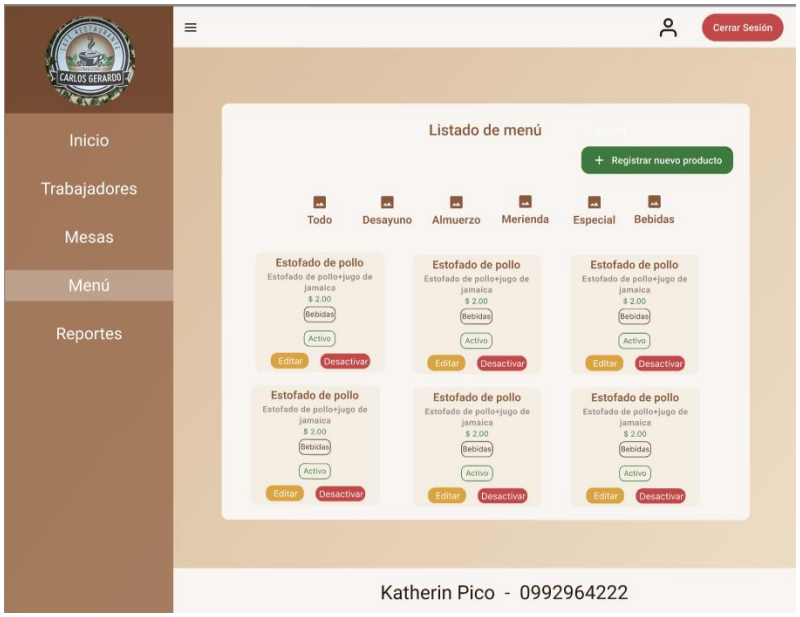
ANEXO D: ITERACIÓN 3 – GESTIÓN DEL MENÚ PLANIFICACIÓN

PLANIFICACIÓN

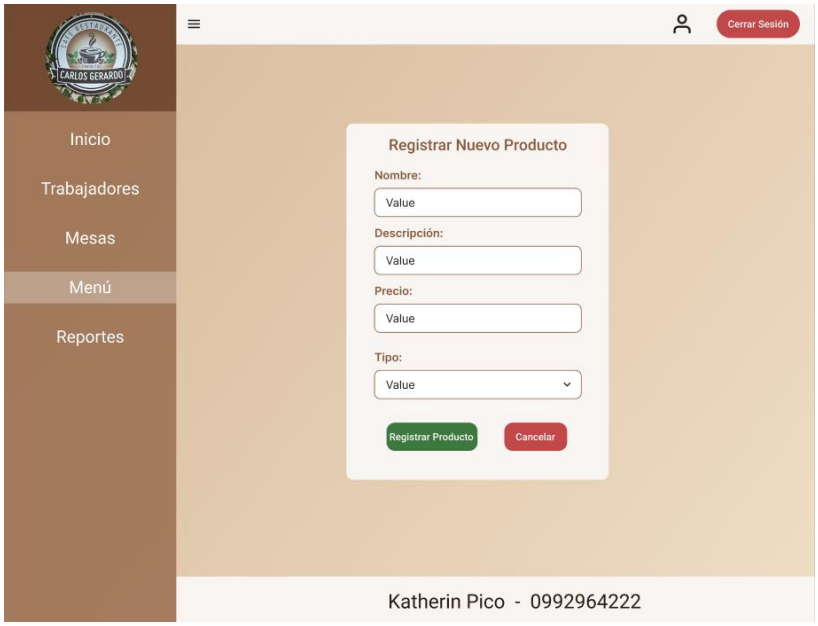
Número	Historia de Usuario
HU06	Registrar productos del menú
HU08	Editar productos del menú
HU09	Desactivar productos del menú

DISEÑO

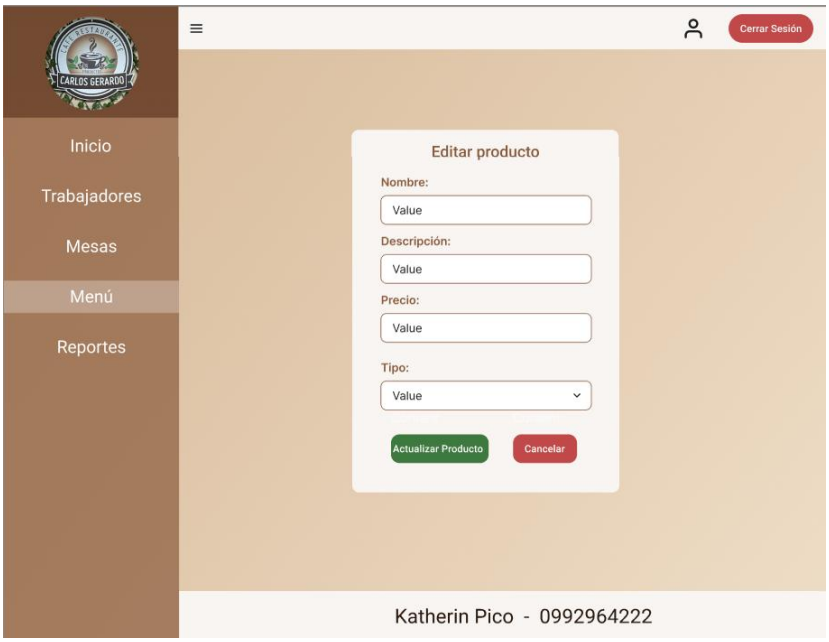
Se observa el listado del menú del restaurante, los cuales ya fueron registrados en el sistema.



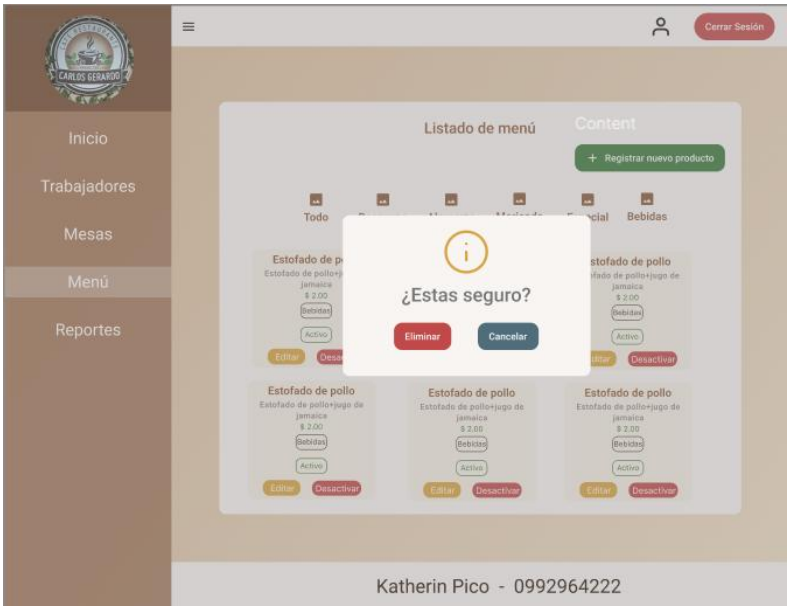
Se observa el formulario de registro de productos para el menú del restaurante en donde se registrará el nombre, una descripción, su precio y el tipo de producto.



Se observa el formulario para editar productos ya registrados en el sistema en caso de existir cambios en la información del producto.



Se observa el mensaje de confirmación para desactivar un producto del listado del menú, ya que, como ese producto corresponde a algunos pedidos, no se puede eliminar, por lo que se optó solo por la desactivación de su información dentro del sistema.



CODIFICACIÓN

En la figura 85 se evidencia el código del template en donde se va a mostrar una lista de los productos del menú registrados en la base de datos.

```
296 <div class="main-content-wrapper menu-page">
297     <div class="container mt-5">
298         <div class="content-card">
342             <div class="row g-4">
343                 {% for p in productos %}
344                 <div class="col-xl-3 col-lg-4 col-md-6 col-sm-6">
345                     <div class="card card-producto p-3 shadow-sm h-100">
346
347                         <h5 class="fw-bold text-center mb-2" style="color: var(--cafe-profundo);">
348                             {{ p.nombre }}
349                         </h5>
350
351                         <p class="text-muted small text-center">
352                             {{ p.descripcion|truncatechars:60 }}
353                         </p>
354
355                         <p class="fw-bold text-success text-center mb-2" style="font-size: 1.1rem;">
356                             ${{ p.precio }}
357                         </p>
358
359                         <div class="text-center mb-2">
360                             <span class="badge
361                                 {% if p.tipo == 'desayuno' %}
362                                     badge-desayuno
363                                 {% elif p.tipo == 'almuerzo' %}
364                                     badge-almuerzo
365                                 {% elif p.tipo == 'merienda' %}
366                                     badge-merienda
367                                 {% elif p.tipo == 'especial' %}
368                                     badge-especial
369                                 {% elif p.tipo == 'bebida' %}
370                                     badge-bebida
371                                 {% endif %}
372                             ">
373                                 {{ p.get_tipo_display }}
374                             </span>
375                         </div>
376
377                         <div class="d-flex flex-column align-items-center gap-2 mt-auto">
378
379                             <!-- ESTADO -->
380                             <div>
381                                 {% if p.activo %}
382                                 <span class="badge bg-activo">Activo</span>
383                                 {% else %}
```

Se evidencia el código del models (modelo), en donde se está creando la tabla de productos para el registro en la base de datos.

```python
# ----------------------------
# Producto / Menú
# ----------------------------
class Producto(models.Model):
    TIPOS = [
        ('desayuno', 'Desayuno'),
        ('almuerzo', 'Almuerzo'),
        ('merienda', 'Merienda'),
        ('especial', 'Especiales'),
        ('bebida', 'Bebidas'),
    ]

    nombre = models.CharField(max_length=100)
    descripcion = models.TextField(blank=True)
    precio = models.DecimalField(max_digits=10, decimal_places=2)
    tipo = models.CharField(max_length=20, choices=TIPOS)
    activo = models.BooleanField(default=True)

    agotado_fecha = models.DateField(blank=True, null=True)
    agotado_hora = models.DateTimeField(blank=True, null=True)

    def __str__(self):
        return self.nombre

    @property
    def agotado_hoy(self):
        return self.agotado_fecha == timezone.localdate()

    @property
    def disponible_hoy(self):
        # activo y NO agotado hoy
        return self.activo and not self.agotado_hoy

    @property
    def tiene_pedidos(self):
        # Usa el related_name por defecto del FK en DetallePedido
        return self.detallepedido_set.exists()
```

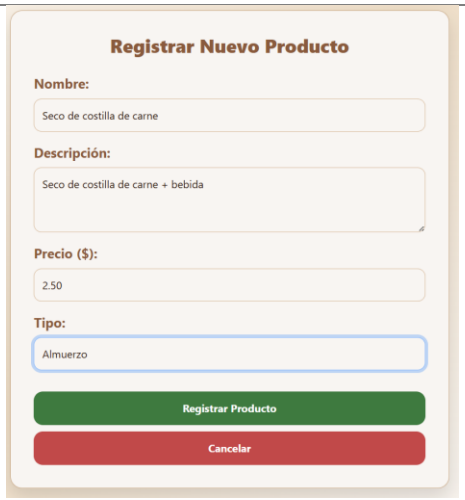
Se observa el código del views en donde se programa la lógica del backend para el CRUD de los productos registrados por el administrador.

```python
# ----------------------------
# Admin - Gestión de Menú / Productos
# ----------------------------
# Listar productos
@login_required(login_url='login')
@rol_requerido('admin')
def listar_menu(request):
    productos = Producto.objects.all().order_by('nombre')
    tipo = request.GET.get("tipo")
    if tipo:
        productos = productos.filter(tipo=tipo)
    return render(request, 'administrador/menu/listar_menu.html', {'productos': productos})

# Registrar producto
@login_required(login_url='login')
@rol_requerido('admin')
def registrar_menu(request):
    if request.method == 'POST':
        nombre = request.POST.get('nombre', '').strip()
        descripcion = request.POST.get('descripcion', '').strip()
        precio = request.POST.get('precio', '').strip()
        tipo = request.POST.get('tipo', '').strip()

        # 1. VALIDAR CAMPOS VACÍOS PRIMERO
        if not nombre or not precio or not tipo:
            messages.error(request, "Todos los campos son obligatorios.")
            return render(request, 'administrador/menu/registrar_menu.html', {
                'nombre': nombre,
                'descripcion': descripcion,
                'precio': precio,
                'tipo': tipo,
            })

        # 2. VALIDAR SOLO LETRAS
        if not re.match(r'^[A-Za-zÁÉÍÓÚáéíóúÑñ ]+$', nombre):
            messages.error(request, "El nombre solo debe contener letras.")
            return render(request, 'administrador/menu/registrar_menu.html', {
                'nombre': nombre,
                'descripcion': descripcion,
                'precio': precio,
                'tipo': tipo,
            })

        # 3. VALIDAR PRECIO
```

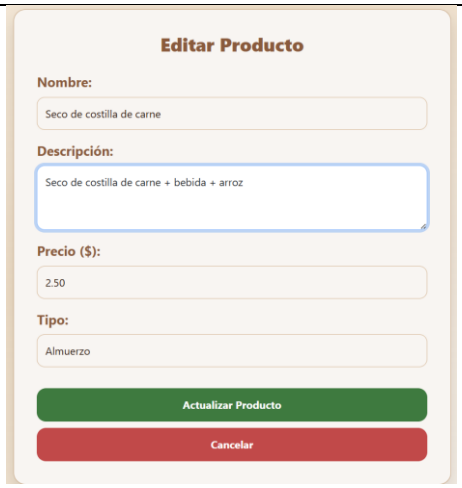
PRUEBAS

Caso de Prueba – HU06 Registrar productos del menú	
Código del Caso de Prueba	CP-11
Historia de Usuario	HU06 – Registrar productos del menú

Nombre del Caso de Prueba	Registrar nuevo producto del menú
Objetivo	Validar que el administrador pueda registrar productos del menú con su precio
Usuario	Administrador
Precondiciones	El administrador debe haber iniciado sesión
Datos de Prueba	Nombre: Seco de costilla de carne Descripción: Seco de costilla de carne + bebida Precio: 2.50 Tipo: Almuerzo
Pasos a Ejecutar	1. Ingresar al sistema como administrador. 2. Ir al módulo Productos. 3. Seleccionar “Registrar producto”. 4. Ingresar los datos del producto. 5. Guardar.
Resultado Esperado	El producto se registra correctamente y aparece en el menú
Resultado Obtenido	El sistema registra correctamente el producto en el menú

Evidencia	 

Caso de Prueba – HU08 Editar productos del menú	
Código del Caso de Prueba	CP-12
Historia de Usuario	HU08 – Editar productos del menú
Nombre del Caso de Prueba	Editar datos de un producto del menú
Objetivo	Validar que el administrador pueda modificar la información de los productos del menú
Usuario	Administrador

Precondiciones	El administrador debe haber iniciado sesión y debe existir al menos un producto registrado
Datos de Prueba	Nombre: Seco de costilla de carne Nueva descripción: Seco de costilla de carne + bebida + papas Precio: 2.50 Tipo: Almuerzo
Pasos a Ejecutar	1. Ingresar al sistema como administrador. 2. Ir al módulo Productos. 3. Buscar el producto a modificar. 4. Seleccionar opción “Editar”. 5. Cambiar los datos. 6. Guardar cambio.
Resultado Esperado	El sistema actualiza correctamente la información del producto y muestra los datos nuevos
Resultado Obtenido	El sistema actualiza correctamente los datos del producto.
Evidencia	 

Caso de Prueba – HU09 Desactivar productos del menú	
Código del Caso de Prueba	CP-13
Historia de Usuario	HU09 – Desactivar productos del menú
Nombre del Caso de Prueba	Desactivar producto del menú
Objetivo	Validar que el administrador pueda desactivar productos que ya no estén disponibles
Usuario	Administrador
Precondiciones	El administrador debe haber iniciado sesión y debe existir al menos un producto activo en el menú
Datos de Prueba	Producto a desactivar: Seco de costilla de carne (estado: activo)
Pasos a Ejecutar	1. Ingresar al sistema como administrador. 2. Ir al módulo Productos. 3. Buscar el producto activo. 4. Seleccionar opción “Desactivar”. 5. Confirmar la acción.
Resultado Esperado	El producto cambia a estado “inactivo” y deja de mostrarse en el menú
Resultado Obtenido	El sistema desactiva correctamente un producto.

Evidencia	

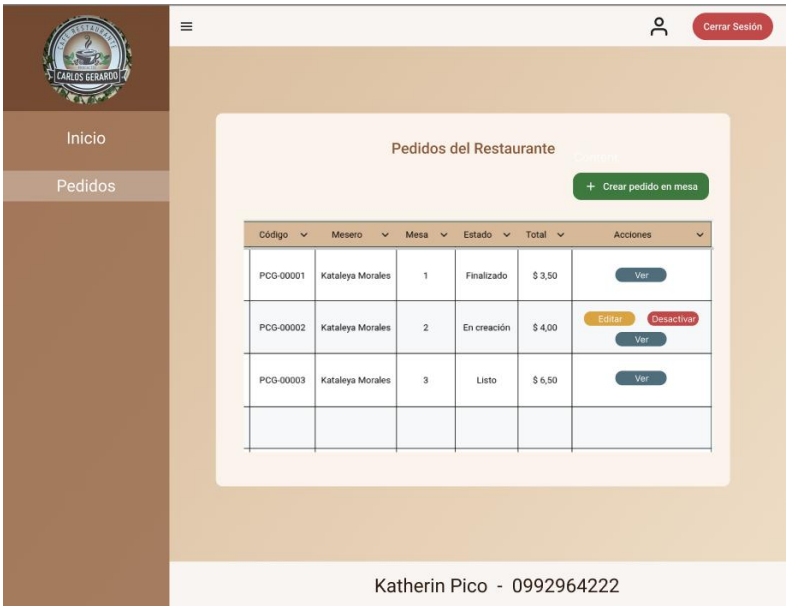
ANEXO E: ITERACIÓN 4 – GESTIÓN DE PEDIDOS EN RESTAURANTE

PLANIFICACIÓN

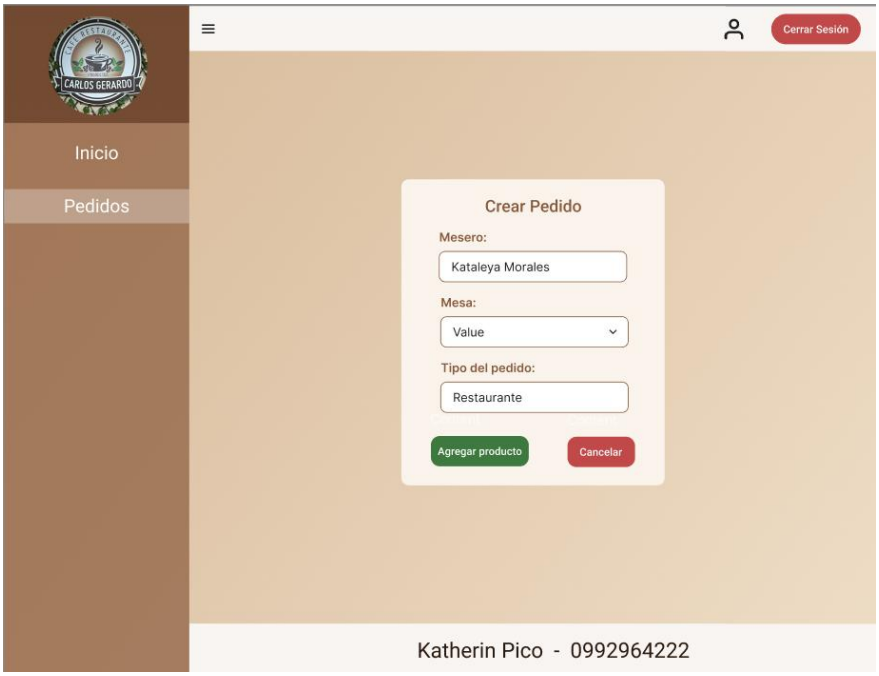
Número	Historia de Usuario
HU14	Crear pedido para mesa
HU15	Agregar productos al pedido
HU16	Editar pedido
HU17	Eliminar pedido
HU18	Ver estado del pedido
HU19	Enviar pedido a cocina

DISEÑO

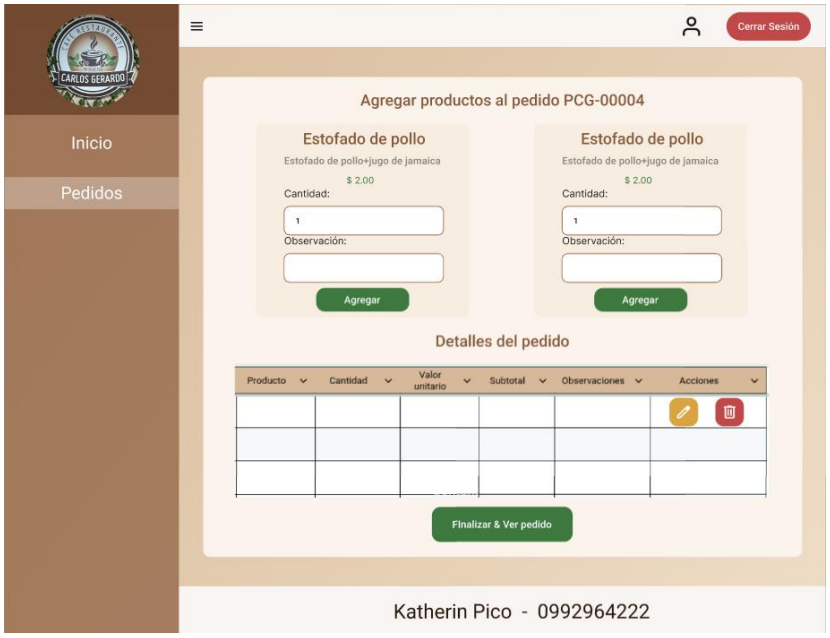
Se observa el listado de los pedidos realizados dentro del restaurante en donde se muestra información relevante del pedido.



Se observa el formulario para crear un pedido dentro del restaurante, en donde los datos que se llenan automáticamente son el nombre del mesero y el tipo de pedido; el único dato a ingresar es el número de mesa.



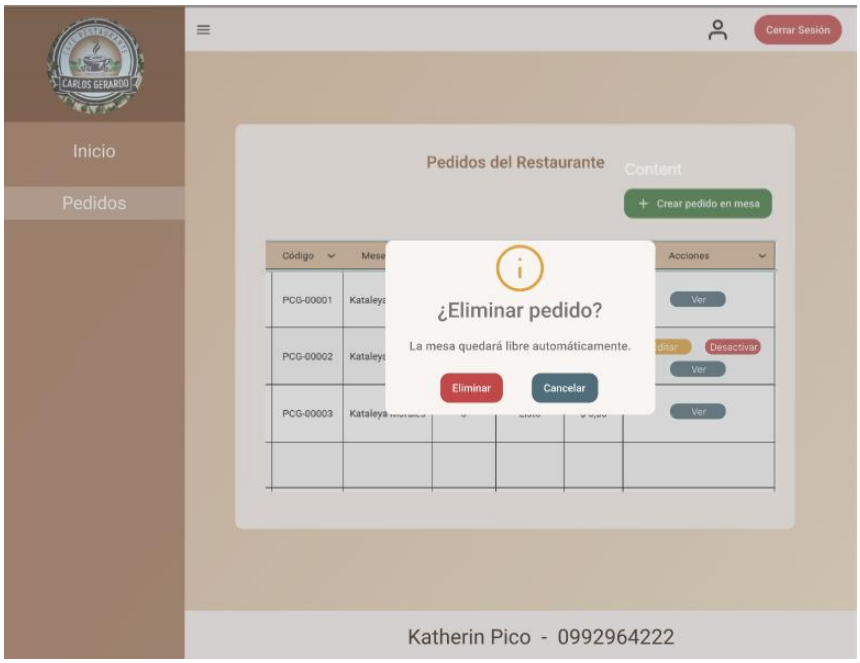
Se observa la interfaz en donde se van a agregar productos al pedido dentro del restaurante, en el cual se registrará la cantidad del producto y una observación en caso de haberla, la cual se mostrará en una lista para corroborar la información con el cliente.



Se observa la interfaz en donde se muestra el detalle del pedido realizado, en el cual se muestra el total del pedido, los productos agregados a dicho pedido, el número de mesa y el mesero encargado del pedido.

Se observa el formulario de editar un pedido dentro del restaurante mientras el estado del pedido está en creación, en el cual se podrá cambiar el número de mesa, los productos agregados, tanto en cantidad como su observación en caso de verla.

Se observa el mensaje de confirmación para eliminar un pedido mientras su estado está en creación; de lo contrario, el pedido no se podrá eliminar.



CODIFICACIÓN

Se evidencia el código del template en donde se va a mostrar una lista de los pedidos del restaurante registrados en la base de datos.

```
<div class="main-content-wrapper">
    <div class="container mt-5">
        <div class="content-card">
            <div class="table-responsive">
                <table id="tbl_pedidos" class="table table-bordered table-striped table-hover align-middle">

                    <td class="text-center estado-pedido" data-id="{{ pedido.id }}">
                    {% if pedido.estado == 'en_creacion' %}
                        <span class="badge badge-creacion">En creación</span>

                    {% elif pedido.estado == 'en preparacion' %}
                        <span class="badge bg-warning">En preparación</span>

                    {% elif pedido.estado == 'listo' %}
                        <span class="badge bg-success">Listo</span>

                    {% elif pedido.estado == 'finalizado' %}
                        <span class="badge bg-primary">Finalizado</span>
                    {% endif %}
                    </td>

                    <td class="text-center">${{ pedido.total }}</td>

                    <td class="text-center acciones-pedido" data-id="{{ pedido.id }}">
                      <a href="{% url 'ver_pedido' pedido.id %}" class="btn btn-info btn-sm">
                        <i class="fas fa-eye"></i> Ver
                      </a>

                      {% if pedido.estado == 'listo' %}
                        <button class="btn btn-fin btn-sm" onclick="finalizarPedido('{{ pedido.id }}')">
                          <i class="fas fa-check"></i> Finalizar
                        </button>
                      {% endif %}

                      {% if pedido.estado == 'en_creacion' %}
                        <a href="{% url 'editar_pedido' pedido.id %}" class="btn btn-warning btn-sm">
                          <i class="fas fa-edit"></i> Editar
                        </a>

                        <button type="button" class="btn btn-danger btn-sm"
                                onclick="confirmarEliminacion('{{ pedido.id }}')">
                          <i class="fas fa-trash"></i> Eliminar
                        </button>
                      {% endif %}
                    </td>
```

Se evidencia el código del models (modelo) en donde se está creando la tabla de pedidos para el registro en la base de datos.

```
class Pedido(models.Model):
    mesero = models.ForeignKey(
        'Usuario',
        on_delete=models.SET_NULL,
        null=True, blank=True,
        related_name='pedidos_mesero',
        limit_choices_to={'rol': 'mesero'}
    )

    cajero = models.ForeignKey(
        'Usuario',
        on_delete=models.SET_NULL,
        null=True, blank=True,
        related_name='pedidos_cajero',
        limit_choices_to={'rol': 'cajero'}
    )


    mesa = models.ForeignKey(
        'Mesa',
        on_delete=models.SET_NULL,
        null=True, blank=True,
        related_name='pedidos'
    )

    estado = models.CharField(max_length=20, choices=ESTADOS, default='en_creacion')
    tipo_pedido = models.CharField(max_length=20, choices=TIPOS)
    nombre_cliente = models.CharField(max_length=150, blank=True, null=True)
    direccion_entrega = models.CharField(max_length=255, blank=True, null=True)
    contacto_cliente = models.CharField(max_length=50, blank=True, null=True)
    fecha_hora = models.DateTimeField(auto_now_add=True)
    subtotal = models.DecimalField(max_digits=10, decimal_places=2, default=0)
    total = models.DecimalField(max_digits=10, decimal_places=2, default=0)
    enviado_cocina = models.BooleanField(default=False)

    codigo_pedido = models.CharField(
        max_length=20,
        unique=True,
        null=True,
        blank=True
    )
```

Se observa el código del views en donde se programa la lógica del backend para el CRUD de los pedidos del restaurante registrados por el mesero.

```python
# ----------------------------
# Mesero-Pedidos
# ----------------------------
@login_required(login_url='login')
@rol_requerido('mesero')
def api_estados_pedidos(request):
    hoy = timezone.localdate()

    pedidos = Pedido.objects.filter(
        tipo_pedido='restaurante',
        mesero=request.user,
        fecha_hora__date=hoy
    ).select_related("mesa").values(
        "id",
        "estado",
        "mesa_id",
        "mesa__estado",
    )

    return JsonResponse(list(pedidos), safe=False)

# Listar pedidos
@login_required(login_url='login')
@rol_requerido('mesero')
def listar_pedidos(request):

    hoy = timezone.localdate()  # Fecha del sistema
    pedidos = Pedido.objects.filter(
        tipo_pedido='restaurante',
        mesero=request.user,
        fecha_hora__date=hoy
    ).order_by('id')

    return render(request, 'mesero/pedidos/listar_pedidos.html', {
        'pedidos': pedidos
    })

# Crear pedido
@login_required(login_url='login')
@rol_requerido('mesero')
def crear_pedido(request):
    if request.method == 'POST':
        mesero_id = request.user.id
        mesa_id = request.POST.get('mesa')
```

PRUEBAS

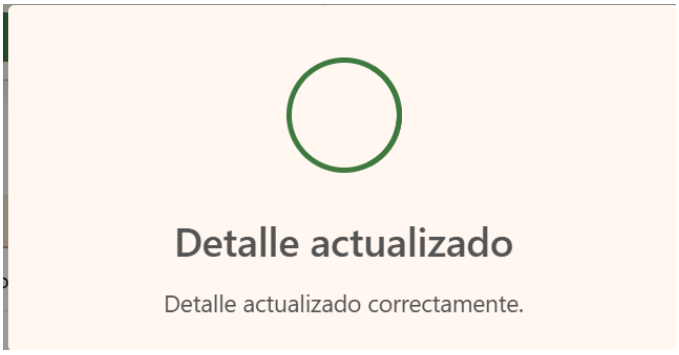
Caso de Prueba – HU14 Crear pedido para mesa	
Código del Caso de Prueba	CP-14

Historia de Usuario	HU14 – Crear pedido para mesa
Nombre del Caso de Prueba	Crear pedido asignado a mesa
Objetivo	Validar que el mesero pueda crear pedidos para mesas
Usuario	Mesero
Precondiciones	El mesero debe haber iniciado sesión y debe existir al menos una mesa disponible
Datos de Prueba	Mesa: 2
Pasos a Ejecutar	1. Ingresar al sistema como mesero. 2. Seleccionar “Nuevo pedido”. 3. Elegir mesa disponible. 4. Guardar pedido.
Resultado Esperado	El pedido se crea correctamente para luego agregar productos.
Resultado Obtenido	El sistema crea un pedido a una mesa seleccionada.
Evidencia	

Caso de Prueba – HU15 Agregar productos al pedido	
Código del Caso de Prueba	CP-15
Historia de Usuario	HU15 – Agregar productos al pedido
Nombre del Caso de Prueba	Agregar productos al pedido
Objetivo	Validar que el mesero agregue productos con cantidad
Usuario	Mesero
Precondiciones	Debe existir un pedido creado
Datos de Prueba	Producto: Petrolero de pollo al jugo Cantidad: 2
Pasos a Ejecutar	1. Ingresar como mesero. 2. Haber creado un pedido para una mesa. 3. Seleccionar producto. 4. Ingresar cantidad. 5. Agregar al pedido.
Resultado Esperado	El producto se agrega correctamente al pedido
Resultado Obtenido	El sistema agrega productos al pedido de forma correcta.

Evidencia	

Caso de Prueba – HU16 Editar pedido	
Código del Caso de Prueba	CP-16
Historia de Usuario	HU16 – Editar pedido
Nombre del Caso de Prueba	Editar productos del pedido
Objetivo	Validar que el mesero pueda modificar el pedido
Usuario	Mesero
Precondiciones	Debe existir un pedido creado y no enviado a cocina

Datos de Prueba	Producto a editar: Seco de pollo al jugo Nueva cantidad: 1
Pasos a Ejecutar	1. Ingresar como mesero. 2. Abrir pedido. 3. Seleccionar producto. 4. Modificar cantidad o eliminar. 5. Guardar cambios.
Resultado Esperado	El pedido se actualiza correctamente
Resultado Obtenido	El sistema edita correctamente un pedido
Evidencia	  

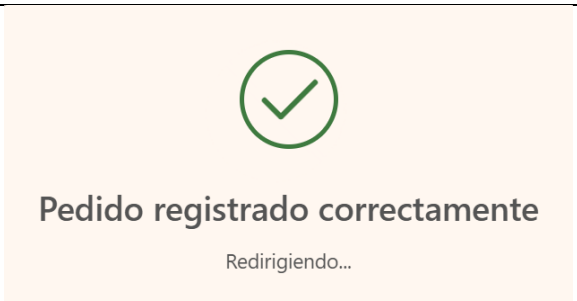
Caso de Prueba – HU17 Eliminar pedido	
Código del Caso de Prueba	CP-17
Historia de Usuario	HU17 – Eliminar pedido
Nombre del Caso de Prueba	Eliminar pedido no preparado
Objetivo	Validar que el mesero elimine pedidos no enviados a cocina
Usuario	Mesero
Precondiciones	Debe existir un pedido en estado “creado”
Datos de Prueba	Pedido: PCG-00006
Pasos a Ejecutar	1. Ingresar como mesero. 2. Buscar pedido. 3. Seleccionar “Eliminar”. 4. Confirmar.
Resultado Esperado	El pedido se elimina correctamente
Resultado Obtenido	El sistema elimino correctamente el pedido de la lista.

Evidencia	

Caso de Prueba – HU18 Ver estado del pedido	
Código del Caso de Prueba	CP-18
Historia de Usuario	HU18 – Ver estado del pedido
Nombre del Caso de Prueba	Visualizar estado del pedido
Objetivo	Verificar que el mesero vea el estado en tiempo real
Usuario	Mesero
Precondiciones	Debe existir al menos un pedido
Datos de Prueba	Pedido: PCG-00005

Pasos a Ejecutar	1. Ingresar como mesero. 2. Abrir módulo pedidos. 3. Visualizar estado.
Resultado Esperado	El sistema muestra el estado real del pedido
Resultado Obtenido	El sistema muestra correctamente el estado del pedido.
Evidencia	  

Caso de Prueba – HU19 Enviar pedido a cocina	
Código del Caso de Prueba	CP-19

Historia de Usuario	HU19 – Enviar pedido a cocina
Nombre del Caso de Prueba	Enviar pedido correctamente
Objetivo	Validar que el pedido llegue a cocina
Usuario	Mesero
Precondiciones	Debe existir pedido con productos agregados
Datos de Prueba	Pedido: PCG-00006
Pasos a Ejecutar	1. Ingresar como mesero. 2. Abrir pedido. 3. Presionar “Enviar a cocina”.
Resultado Esperado	El pedido cambia a estado “pendiente en cocina”
Resultado Obtenido	El sistema envía correctamente el pedido a cocina.
Evidencia	 

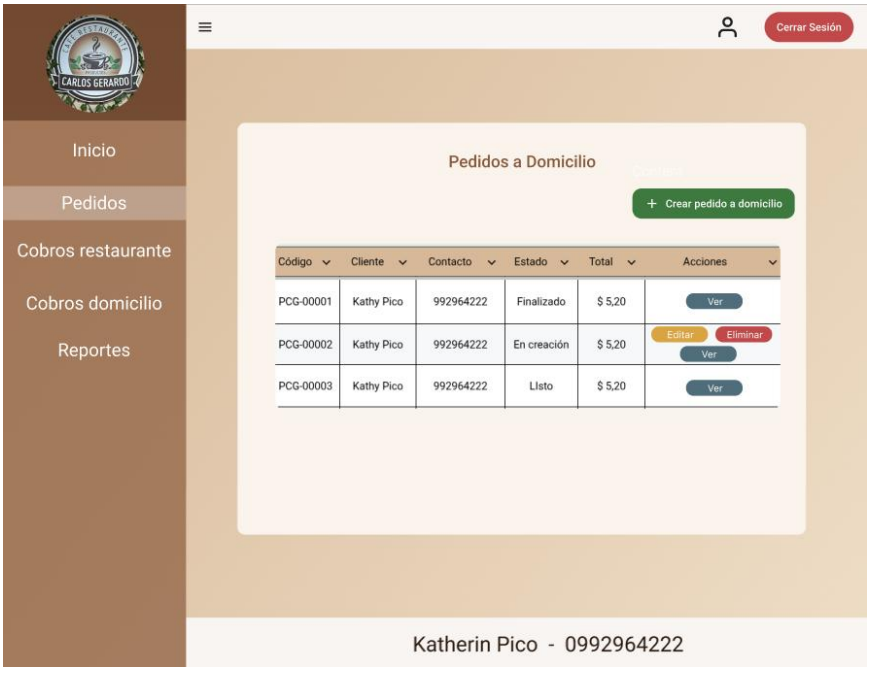
ANEXO F: ITERACIÓN 5 – GESTIÓN DE PEDIDOS A DOMICILIO

PLANIFICACIÓN

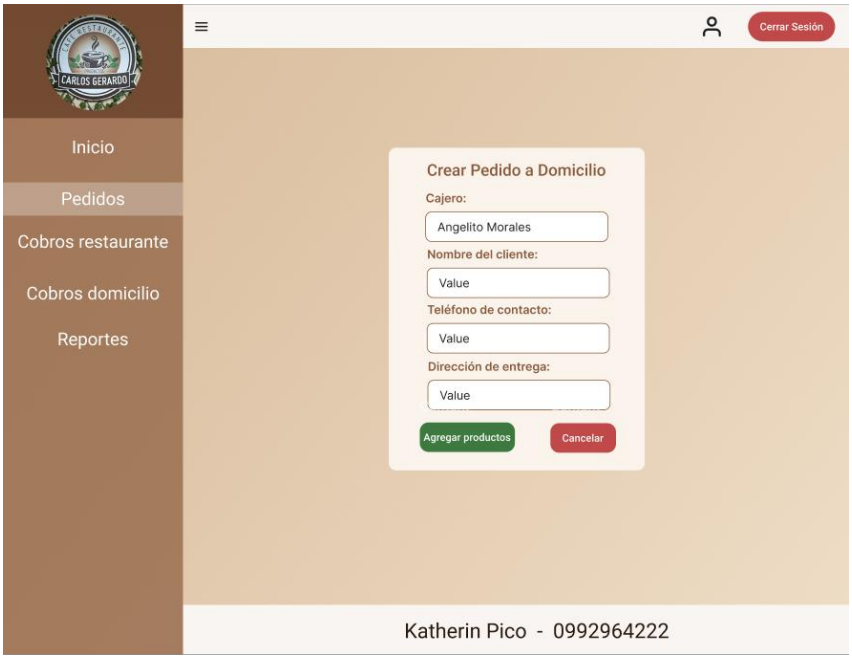
Número	Historia de Usuario
HU26	Registrar pedido a domicilio
HU35	Visualizar pedidos a domicilio
HU36	Agregar productos a pedido a domicilio
HU37	Editar pedidos a domicilio
HU38	Eliminar pedidos a domicilio
HU39	Enviar pedidos a domicilio a cocina
HU40	Visualizar estado de pedidos a domicilio

DISEÑO

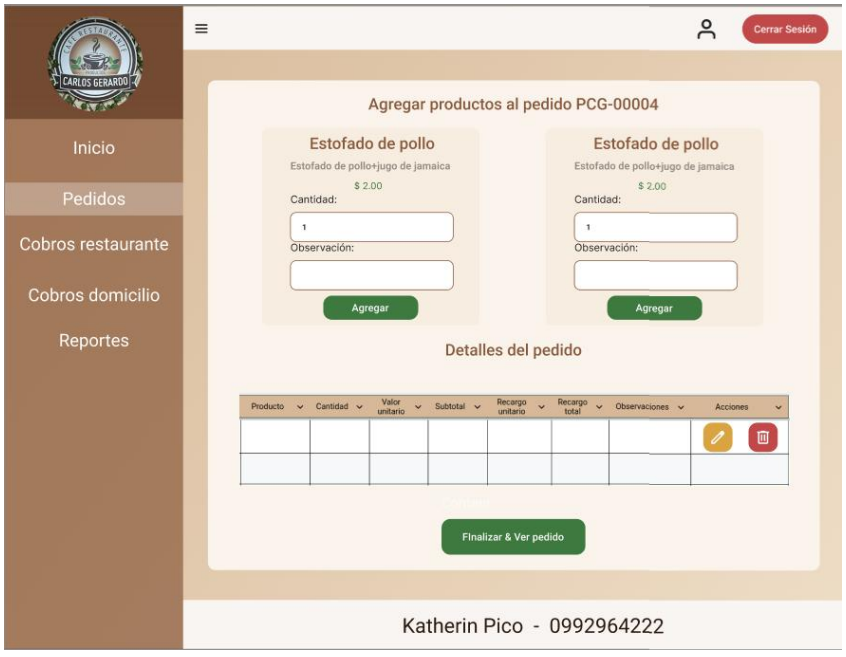
Se observa el listado de los pedidos a domicilio que realizo el cajero en donde se muestra información relevante del pedido.



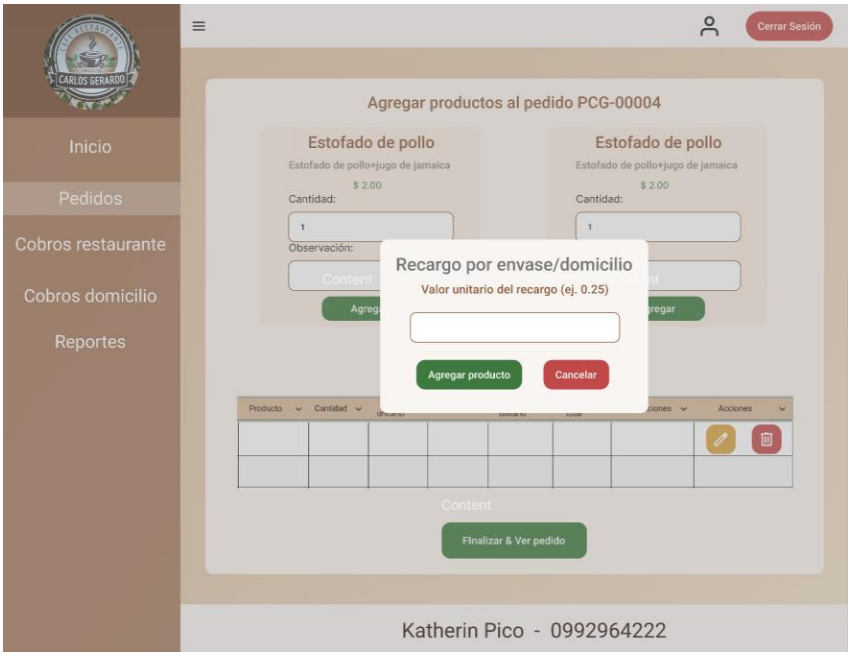
Se observa el formulario para crear pedidos a domicilio, en donde los datos que se llenan automáticamente son el nombre del cajero y los datos a ingresar son el nombre, el teléfono y la dirección del cliente.



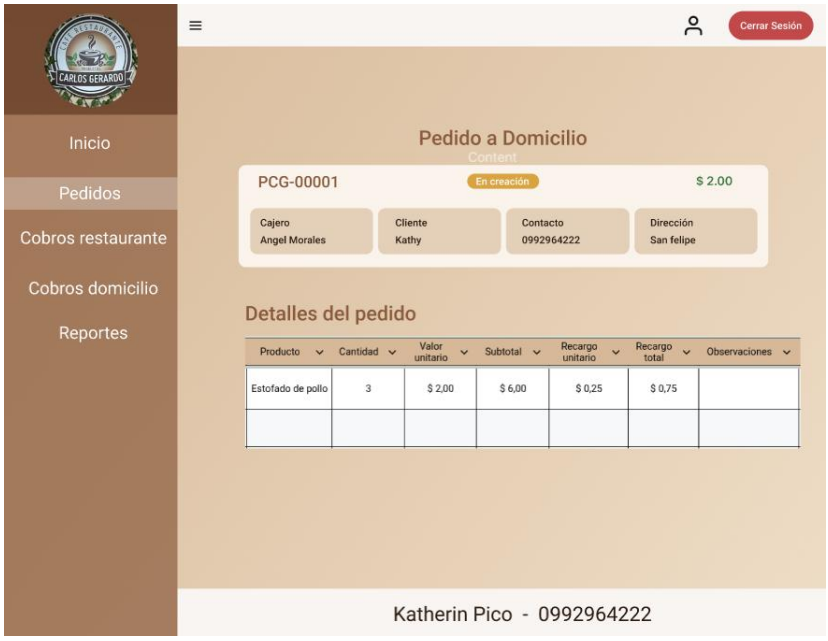
Se observa la interfaz en donde se van a agregar productos al pedido a domicilio, en el cual se registrará la cantidad del producto y una observación en caso de haberla, para luego ser mostrada en una lista para corroborar la información con el cliente.



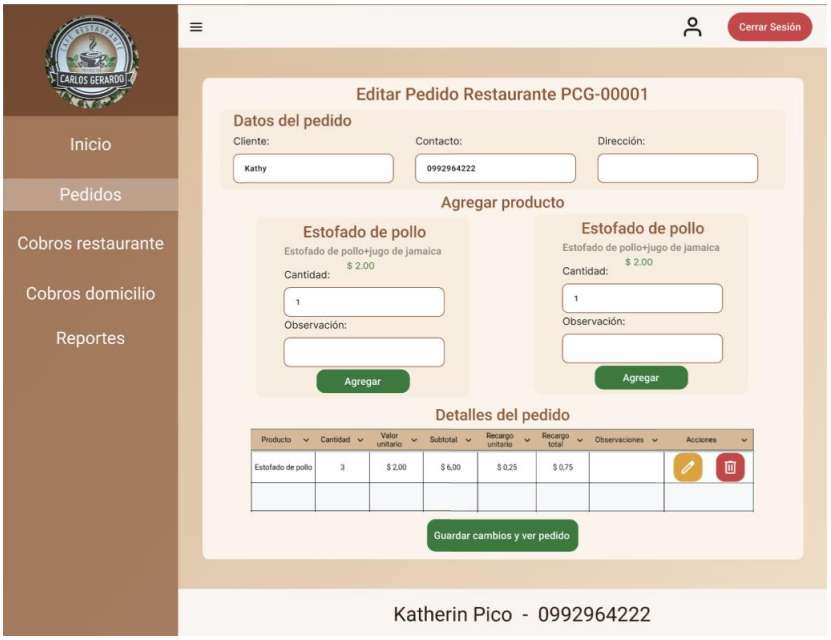
Se observa un pequeño SweetAlert para ingresar el valor del recargo por los envases de la comida.



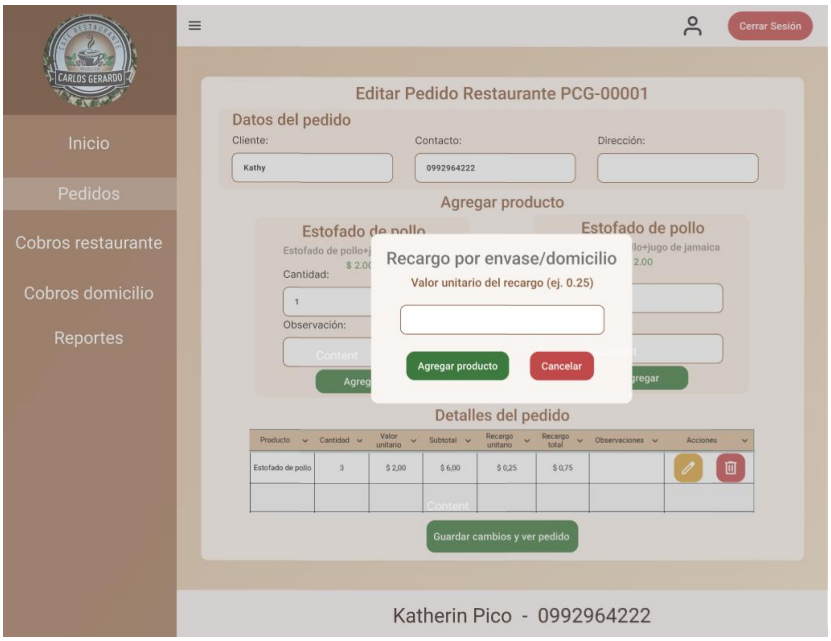
Se observa la información del pedido registrado, en el cual se muestra el nombre del cajero, la información del cliente para realizar la entrega del pedido y los productos que pertenecen a dicho pedido de la comida.



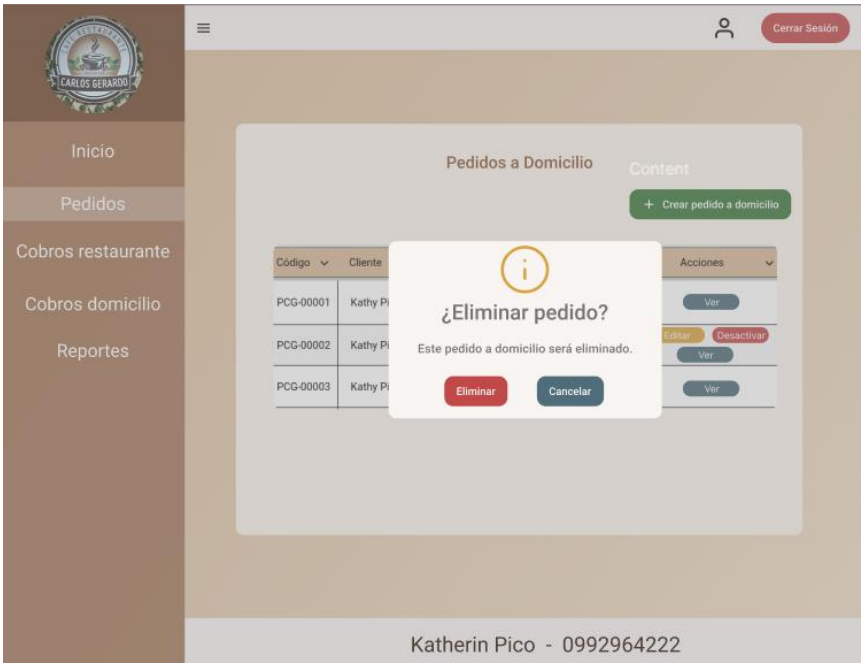
Se observa el formulario de editar un pedido a domicilio mientras el estado del pedido está en creación, en el cual se podrá cambiar la información del cliente, los productos agregados, tanto en cantidad como su observación en caso de verla.



Se observa un pequeño SweetAlert para ingresar el valor del recargo por los envases de la comida en el formulario de editar el pedido.



Se observa un mensaje de confirmación para eliminar un pedido a domicilio mientras su estado está en creación; de lo contrario, no se podrá eliminar.



CODIFICACIÓN

Se evidencia el código del template en donde se va a mostrar una lista de los pedidos a domicilio registrados en la base de datos.

```
<div class="main-content-wrapper">
    <div class="container mt-5">
        <div class="content-card">
            <div class="table-responsive">
                <table class="table table-bordered table-striped table-hover" id="tbl_pedidos">

                    <tbody>
                        {% for pedido in pedidos %}
                        <tr data-pedido-row="{{ pedido.id }}">

                            <td>{{ pedido.codigo_pedido }}</td>
                            <td>{{ pedido.nombre_cliente }}</td>
                            <td>{{ pedido.contacto_cliente }}</td>

                            <td class="estado-pedido-domicilio" data-id="{{ pedido.id }}">
                                {% if pedido.estado == 'en_creacion' %}
                                <span class="badge badge-creacion">En creación</span>
                                {% elif pedido.estado == 'en preparacion' %}
                                <span class="badge bg-warning">En preparación</span>
                                {% elif pedido.estado == 'listo' %}
                                <span class="badge bg-success">Listo</span>
                                {% elif pedido.estado == 'finalizado' %}
                                <span class="badge bg-primary">Finalizado</span>
                                {% endif %}
                            </td>

                            <td>$ <span class="money" data-valor="{{ pedido.total }}">{{ pedido.total }}</span></td>

                            <td class="acciones-pedido-domicilio" data-id="{{ pedido.id }}">
                                <a href="{% url 'cajero_ver_pedido' pedido.id %}" class="btn btn-info btn-sm">
                                    <i class="fas fa-eye"></i> Ver
                                </a>

                                {% if pedido.estado == 'en_creacion' %}
                                <a href="{% url 'cajero_editar_pedido' pedido.id %}" class="btn btn-warning btn-sm">
                                    <i class="fas fa-edit"></i> Editar
                                </a>

                                <button class="btn btn-danger btn-sm"
                                    onclick="confirmarEliminarDomicilio('{{ pedido.id }}')">
                                    <i class="fas fa-trash"></i> Eliminar
                                </button>
                                {% endif %}
                            </td>
```

Se evidencia el código del models (modelo), en donde se está creando la tabla de pedidos para el registro en la base de datos.

```python
class Pedido(models.Model):
    mesero = models.ForeignKey(
        'Usuario',
        on_delete=models.SET_NULL,
        null=True, blank=True,
        related_name='pedidos_mesero',
        limit_choices_to={'rol': 'mesero'}
    )

    cajero = models.ForeignKey(
        'Usuario',
        on_delete=models.SET_NULL,
        null=True, blank=True,
        related_name='pedidos_cajero',
        limit_choices_to={'rol': 'cajero'}
    )


    mesa = models.ForeignKey(
        'Mesa',
        on_delete=models.SET_NULL,
        null=True, blank=True,
        related_name='pedidos'
    )

    estado = models.CharField(max_length=20, choices=ESTADOS, default='en_creacion')
    tipo_pedido = models.CharField(max_length=20, choices=TIPOS)
    nombre_cliente = models.CharField(max_length=150, blank=True, null=True)
    direccion_entrega = models.CharField(max_length=255, blank=True, null=True)
    contacto_cliente = models.CharField(max_length=50, blank=True, null=True)
    fecha_hora = models.DateTimeField(auto_now_add=True)
    subtotal = models.DecimalField(max_digits=10, decimal_places=2, default=0)
    total = models.DecimalField(max_digits=10, decimal_places=2, default=0)
    enviado_cocina = models.BooleanField(default=False)

    codigo_pedido = models.CharField(
        max_length=20,
        unique=True,
        null=True,
        blank=True
    )
```

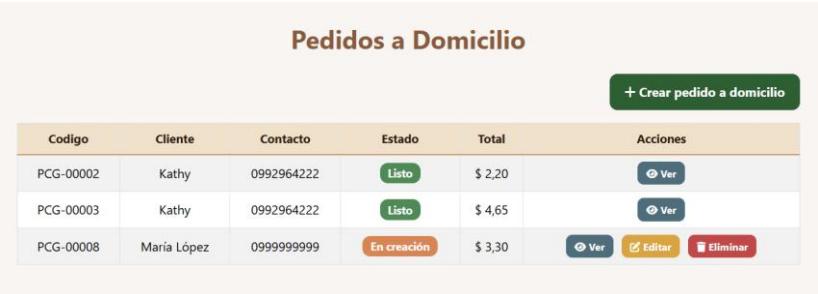
Se observa el código del views en donde se programa la lógica del backend para el CRUD de los pedidos a domicilio registrados por el cajero.

```python
# ----------------------------
# Cajero - Pedidos a domicilio
# ----------------------------
@login_required(login_url='login')
@rol_requerido('cajero')
def cajero_listar_pedidos(request):
    hoy = timezone.localdate()

    pedidos = Pedido.objects.filter(
        tipo_pedido='domicilio',
        cajero=request.user,
        fecha_hora__date=hoy
    ).exclude(estado='borrador').order_by('id')


    return render(request, 'cajero/pedidos/listar_pedidos.html', {'pedidos': pedidos})

@login_required(login_url='login')
@rol_requerido('cajero')
def cajero_crear_pedido(request):
    """
    Crea el pedido a domicilio: SOLO datos del cliente.
    El recargo por producto se pedirá al agregar cada detalle.
    """
    if request.method == 'POST':
        nombre_cliente = request.POST.get('nombre_cliente', '').strip()
        contacto_cliente = request.POST.get('contacto_cliente', '').strip()
        direccion_entrega = request.POST.get('direccion_entrega', '').strip()

        # Validaciones básicas
        if not nombre_cliente or not contacto_cliente or not direccion_entrega:
            messages.error(request, "Todos los datos del cliente son obligatorios.")
            return render(request, 'cajero/pedidos/crear_pedido.html', {
                'user': request.user,
                'nombre_cliente': nombre_cliente,
                'contacto_cliente': contacto_cliente,
                'direccion_entrega': direccion_entrega,
            })

        # Crear pedido a domicilio (sin recargo, se usará recargo por detalle)
        pedido = Pedido.objects.create(
            cajero=request.user,
            tipo_pedido='domicilio',
            estado='borrador',
            nombre_cliente=nombre_cliente,
```

PRUEBAS

Caso de Prueba – HU26 Registrar pedido a domicilio	
Código del Caso de Prueba	CP-20
Historia de Usuario	HU26 – Registrar pedido a domicilio

Nombre del Caso de Prueba	Registrar pedido a domicilio
Objetivo	Validar que el cajero registre pedidos a domicilio con datos del cliente
Usuario	Cajero
Precondiciones	El cajero debe haber iniciado sesión
Datos de Prueba	Cliente: María López Teléfono: 0999999999 Dirección: Av. Central y Loja
Pasos a Ejecutar	1. Ingresar al sistema como cajero. 2. Seleccionar “Nuevo pedido a domicilio”. 3. Ingresar datos del cliente. 4. Guardar pedido.
Resultado Esperado	El pedido a domicilio se registra correctamente
Resultado Obtenido	El sistema registra los datos del cliente para el pedido a domicilio
Evidencia	

Caso de Prueba – HU35 Visualizar pedidos a domicilio	
Código del Caso de Prueba	CP-21
Historia de Usuario	HU35 – Visualizar pedidos a domicilio
Nombre del Caso de Prueba	Visualizar listado de pedidos a domicilio
Objetivo	Validar que el cajero pueda ver los pedidos a domicilio registrados
Usuario	Cajero
Precondiciones	El cajero debe haber iniciado sesión y deben existir pedidos a domicilio registrados
Datos de Prueba	Pedido: PCG-00002 Pedido: PCG-00003 Pedido: PCG-00008
Pasos a Ejecutar	1. Ingresar al sistema como cajero. 2. Ir al módulo Pedidos a domicilio. 3. Seleccionar opción “Listar pedidos”.
Resultado Esperado	El sistema muestra los pedidos a domicilio con datos del cliente y estado
Resultado Obtenido	El sistema muestra correctamente los datos del pedido en una tabla.
Evidencia	

Caso de Prueba – HU36 Agregar productos a pedido a domicilio	
Código del Caso de Prueba	CP-22
Historia de Usuario	HU36 – Agregar productos a pedido a domicilio
Nombre del Caso de Prueba	Agregar productos a pedido a domicilio
Objetivo	Validar que el cajero agregue productos al pedido a domicilio
Usuario	Cajero
Precondiciones	Debe existir un pedido a domicilio creado
Datos de Prueba	Producto: Seco de pollo al jugo Cantidad: 3
Pasos a Ejecutar	1. Ingresar como cajero. 2. Abrir pedido a domicilio. 3. Seleccionar producto. 4. Ingresar cantidad. 5. Agregar.
Resultado Esperado	El producto se agrega correctamente al pedido
Resultado Obtenido	El sistema agrego correctamente un producto al pedido creado

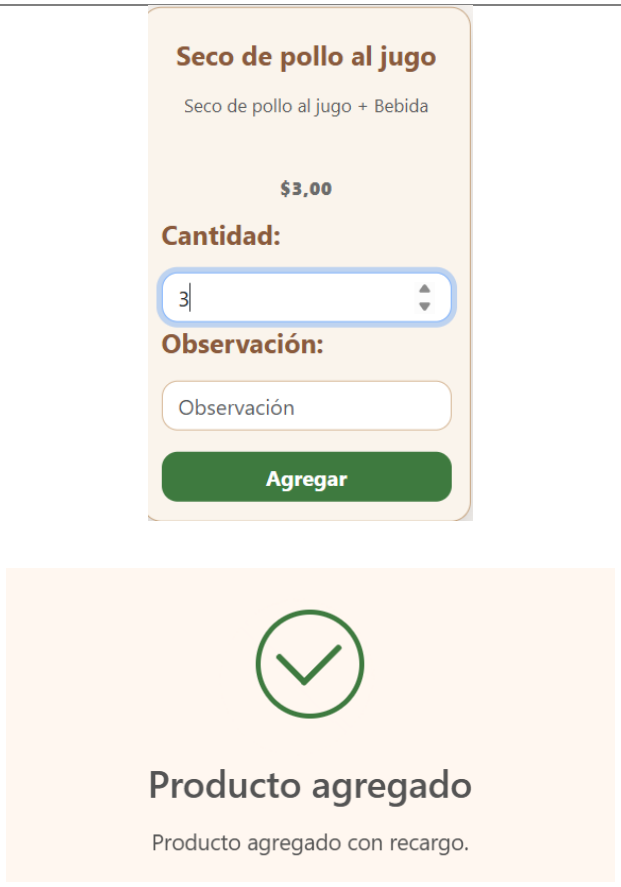
Evidencia	

Figura 8.1. Caso de prueba: Editar pedido a domicilio.

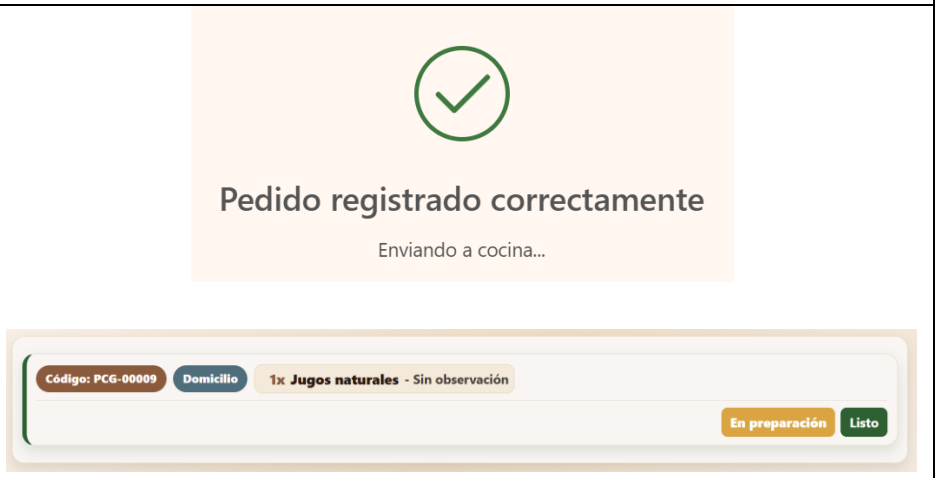
Caso de Prueba – HU37 Editar pedidos a domicilio	
Código del Caso de Prueba	CP-23
Historia de Usuario	HU37 – Editar pedidos a domicilio
Nombre del Caso de Prueba	Editar pedido a domicilio
Objetivo	Validar que el cajero modifique pedidos antes de enviarlos a cocina
Usuario	Cajero
Precondiciones	Debe existir un pedido a domicilio no enviado a cocina

Datos de Prueba	Producto: Seco de pollo al jugo Nueva cantidad: 2
Pasos a Ejecutar	1. Ingresar como cajero. 2. Abrir pedido a domicilio. 3. Editar producto o cantidad. 4. Guardar cambios.
Resultado Esperado	El pedido se actualiza correctamente
Resultado Obtenido	El sistema actualiza correctamente los datos del pedido.
Evidencia	

Caso de Prueba – HU38 Eliminar pedidos a domicilio	
Código del Caso de Prueba	CP-24
Historia de Usuario	HU38 – Eliminar pedidos a domicilio
Nombre del Caso de Prueba	Eliminar pedido a domicilio no preparado
Objetivo	Validar que el cajero elimine pedidos no enviados a cocina
Usuario	Cajero
Precondiciones	Debe existir pedido a domicilio en estado “creado”
Datos de Prueba	Producto: PCG-00008
Pasos a Ejecutar	1. Ingresar como cajero. 2. Abrir pedido a domicilio. 3. Seleccionar producto. 4. Ingresar cantidad. 5. Agregar.
Resultado Esperado	El producto se agrega correctamente al pedido
Resultado Obtenido	El sistema agrego correctamente un producto al pedido creado

Evidencia	

Caso de Prueba – HU38 Eliminar pedidos a domicilio	
Código del Caso de Prueba	CP-25
Historia de Usuario	HU39 – Enviar pedidos a domicilio a cocina
Nombre del Caso de Prueba	Enviar pedido a domicilio a cocina
Objetivo	Validar que el pedido llegue a cocina
Usuario	Cajero
Precondiciones	Debe existir pedido a domicilio con productos
Datos de Prueba	Producto: PCG-00008
Pasos a Ejecutar	1. Ingresar como cajero. 2. Abrir pedido. 3. Presionar “Enviar a cocina”.
Resultado Esperado	El pedido cambia a estado “pendiente en cocina”

Resultado Obtenido	El sistema envía el pedido a domicilio a cocina
Evidencia	

Caso de Prueba – HU40 Visualizar estado de pedidos a domicilio	
Código del Caso de Prueba	CP-26
Historia de Usuario	HU40 – Visualizar estado de pedidos a domicilio
Nombre del Caso de Prueba	Visualizar estado del pedido a domicilio
Objetivo	Validar que el cajero vea el estado en tiempo real
Usuario	Cajero
Precondiciones	Debe existir al menos un pedido a domicilio
Datos de Prueba	Producto: PCG-00010
Pasos a Ejecutar	1. Ingresar como cajero. 2. Ir a pedidos a domicilio. 3. Ver estado del pedido.

Resultado Esperado	El sistema muestra el estado real del pedido
Resultado Obtenido	El sistema muestra el estado de los pedidos a domicilio.
Evidencia	   

ANEXO G: ITERACIÓN 6 – GESTIÓN DE COCINA

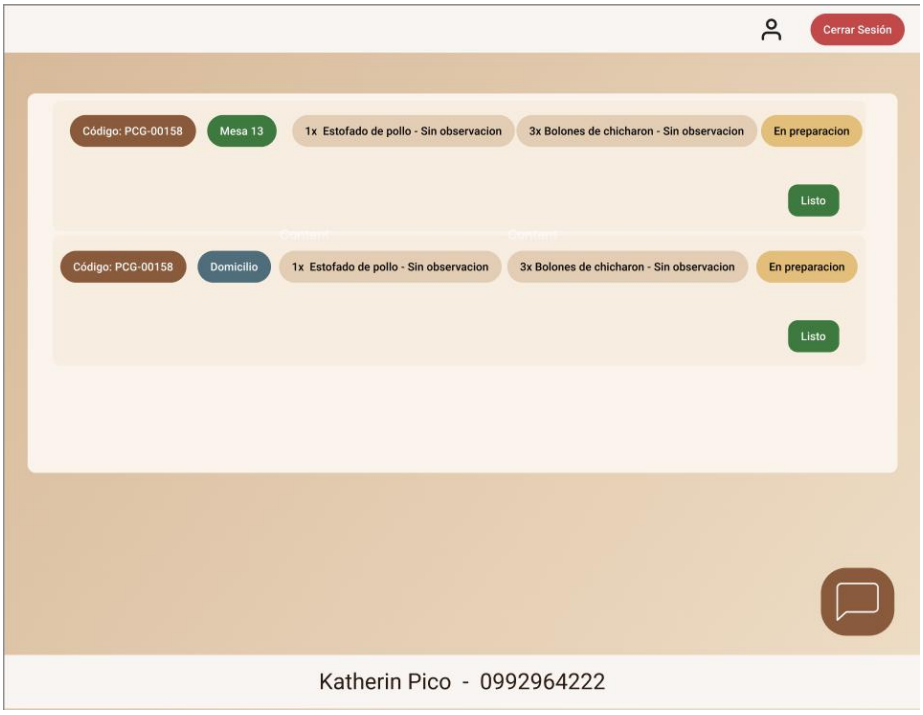
PLANIFICACIÓN

Número	Historia de Usuario
HU20	Visualizar pedidos pendientes
HU21	Marcar pedido en preparación
HU22	Marcar pedido como listo
HU23	Avisar producto agotado

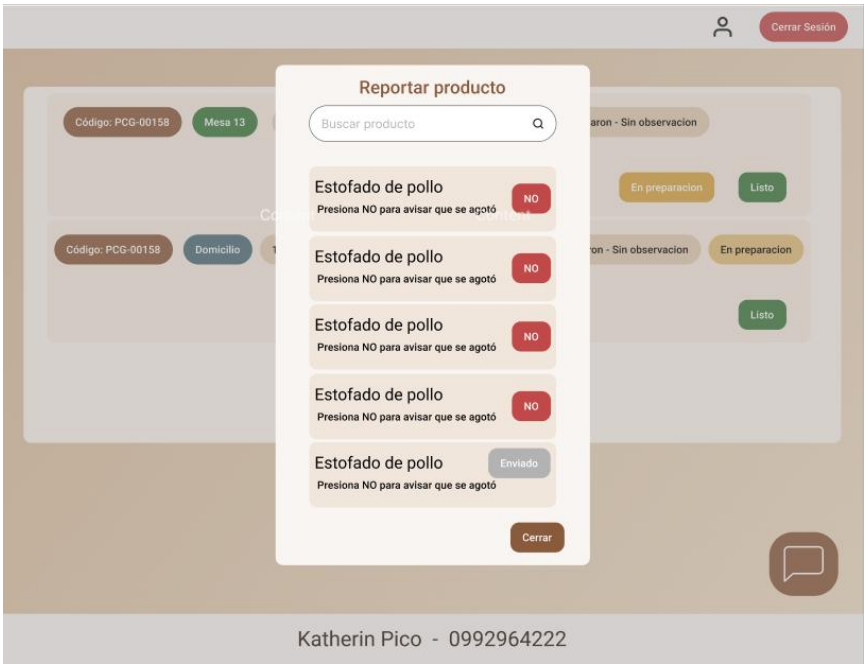
DISEÑO

Se observa la información del pedido realizado dentro del restaurante y los que son a domicilio para que los cocineros empiecen a preparar el producto.

Se observa que los pedidos enviados a cocina ya fueron marcados en preparación, es decir, que los cocineros ya están realizando el pedido para la entrega al cliente.



Se observa el SweetAlert para que los cocineros informen a los meseros y al cajero que un producto del menú ya se agotó.



CODIFICACIÓN

Se evidencia el código del template en donde se va a mostrar una lista de los pedidos a preparar registrados en la base de datos.

```
<div class="main-content-wrapper">
    <div class="container mt-4">
        <div class="content-card">
            <div id="pedidos-cocina" class="pedidos-lista">
                {% for pedido in pedidos_restaurante %}
                <div id="pedido-{{ pedido.id }}" data-pedido-id="{{ pedido.id }}">
                    <div class="pedido-card
                        {% if pedido.estado == 'en preparacion' %}estado-preparacion{% else %}estado-nuevo{% endif %}">

                        {% if pedido.estado != 'en preparacion' %}
                            <span class="badge-estado">En preparación</span>
                        {% endif %}

                        <div class="pedido-topline">
                            <div class="pedido-meta">
                                <span class="pill pill-codigo">Código: {{ pedido.codigo_pedido }}</span>
                                <span class="pill pill-mesa">Mesa {{ pedido.mesa.numero }}</span>
                            </div>

                            <div class="pedido-productos-inline">
                                {% for d in pedido.detalles.all %}
                                <span class="prod-inline">
                                    <span class="qty">{{ d.cantidad }}x</span>
                                    <span class="nombre">{{ d.producto.nombre }}</span>
                                    <span class="obs">- {{ d.observacion|default:"Sin observación" }}</span>
                                </span>
                                {% endfor %}
                            </div>
                        </div>

                        <div class="d-flex justify-content-end gap-2 mt-3">
                            {% if pedido.estado != 'en preparacion' %}
                            <button class="btn btn-warning btn-preparacion btn-marcar-preparacion"
                                    data-pedido-id="{{ pedido.id }}">
                                En preparación
                            </button>
                            {% endif %}

                            <button class="btn btn-success btn-marcar-listo" data-pedido-id="{{ pedido.id }}">
                                Listo
                            </button>
                        </div>
                    </div>
```

Se evidencia el código del models (modelo), en donde se está creando la tabla de pedidos para el registro en la base de datos.

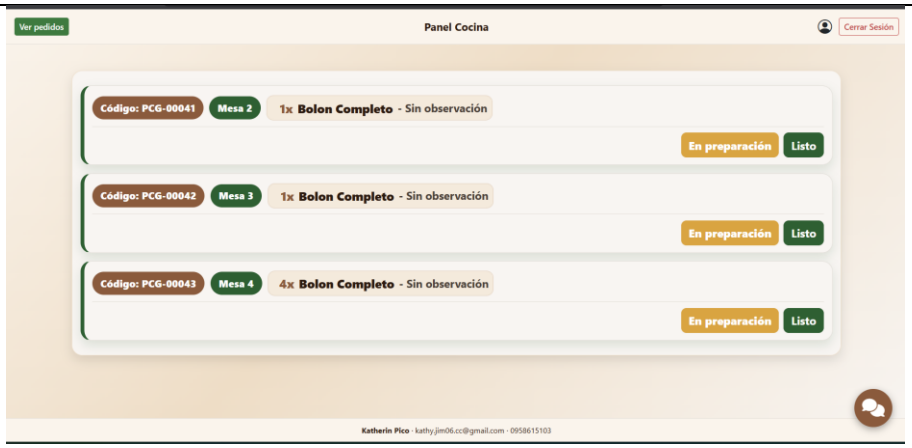
```python
# ----------------------------
# Detalle de pedido
# ----------------------------
class DetallePedido(models.Model):
    pedido = models.ForeignKey(Pedido, on_delete=models.CASCADE, related_name='detalles')
    producto = models.ForeignKey('Producto', on_delete=models.CASCADE)
    cantidad = models.PositiveIntegerField()
    precio_unitario = models.DecimalField(max_digits=10, decimal_places=2)
    subtotal = models.DecimalField(max_digits=10, decimal_places=2, default=0)

    # NUEVO CAMPO
    observacion = models.CharField(max_length=200, blank=True, null=True)
    recargo = models.DecimalField(max_digits=10, decimal_places=2, default=0)  # <-- nuev


    def save(self, *args, **kwargs):
        # Asigna precio actual del producto si no se define
        if not self.precio_unitario:
            self.precio_unitario = self.producto.precio

        # calcula subtotal
        self.subtotal = self.cantidad * self.precio_unitario

        super().save(*args, **kwargs)

        # actualizar subtotal total del pedido
        self.pedido.calcular_totales()

    @property
    def recargo_total(self):
        return self.recargo * self.cantidad


    @property
    def total_con_recargo(self):
        return self.subtotal + self.recargo_total

    def __str__(self):
        return f"{self.cantidad} x {self.producto.nombre}"
```

Se observa el código del views en donde se programa la lógica del backend para la lista de pedidos a preparar por cocina registrados por el cajero y mesero.

```python
@login_required(login_url='login')
@rol_requerido('cocinero')
def vista_cocina(request):
    hoy = timezone.localdate()
    estados = ["en_creacion", "en preparacion"]

    pedidos_restaurante = Pedido.objects.filter(
        estado__in=estados,
        tipo_pedido='restaurante',
        fecha_hora__date=hoy
    ).order_by('id')

    pedidos_domicilio = Pedido.objects.filter(
        estado__in=estados,
        tipo_pedido='domicilio',
        fecha_hora__date=hoy
    ).order_by('id')

    meseros = Usuario.objects.filter(rol='mesero')
    cajeros = Usuario.objects.filter(rol='cajero')

    # NUEVO: menú
    productos_menu = Producto.objects.filter(activo=True).order_by('nombre')  #

    return render(request, 'cocinero/pedido.html', {
        'pedidos_restaurante': pedidos_restaurante,
        'pedidos_domicilio': pedidos_domicilio,
        'meseros': meseros,
        'cajeros': cajeros,
        'productos_menu': productos_menu,  # NUEVO
    })


@login_required(login_url='login')
@rol_requerido('cocinero')
@csrf_protect
@require_POST
def marcar_pedido_listo(request, pedido_id):
    pedido = get_object_or_404(Pedido, id=pedido_id)

    if pedido.estado != "en preparacion":
        return JsonResponse({"success": False, "mensaje": "Solo se puede marcar

    pedido.estado = "listo"
```

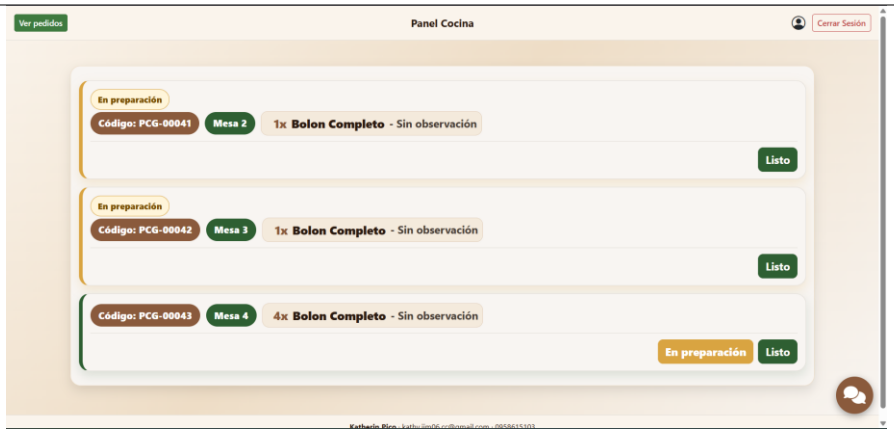
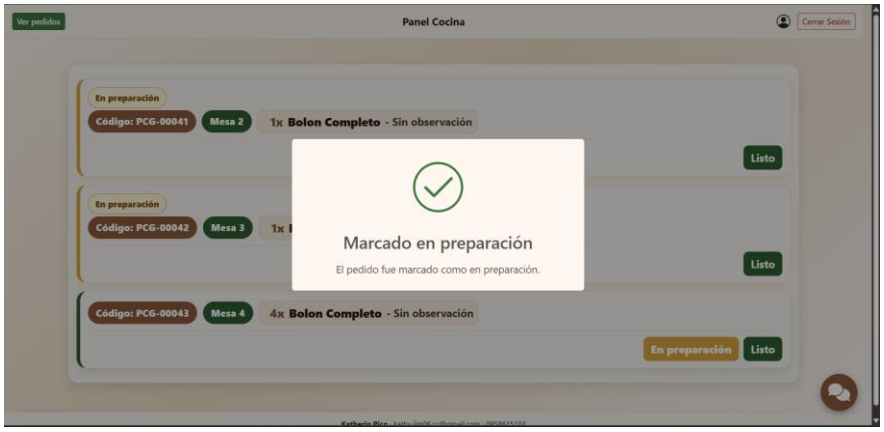
PRUEBAS

Caso de Prueba – HU20 Visualizar pedidos pendientes	
Código del Caso de Prueba	CP-27
Historia de Usuario	HU20 – Visualizar pedidos pendientes
Nombre del Caso de Prueba	Visualizar pedidos pendientes

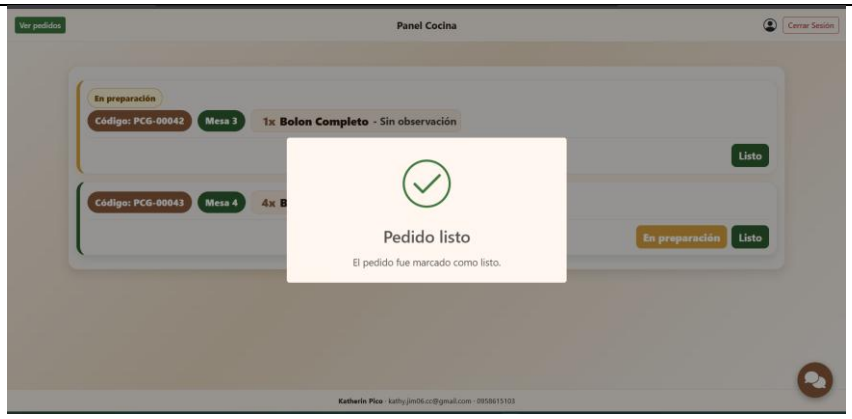
Objetivo	Validar que el cocinero pueda visualizar los pedidos pendientes del día para preparar los platos en orden.
Usuario	Cocinero
Precondiciones	El cocinero debe haber iniciado sesión en el sistema.
Datos de Prueba	Pedido N°: 105 Producto: Arroz con pollo Estado: Pendiente
Pasos a Ejecutar	1. Ingresar al sistema como cocinero. 2. Acceder al módulo de pedidos pendientes. 3. Visualizar la lista de pedidos del día.
Resultado Esperado	El sistema muestra correctamente la lista de pedidos pendientes del día.
Resultado Obtenido	El sistema muestra los pedidos pendientes registrados para su preparación.
Evidencia	

Caso de Prueba – HU21 Marcar pedido en preparación	
Código del Caso de Prueba	CP-28

Historia de Usuario	HU21 – Marcar pedido en preparación
Nombre del Caso de Prueba	Marcar pedido en preparación
Objetivo	Validar que el cocinero pueda cambiar el estado de un pedido a "En preparación".
Usuario	Cocinero
Precondiciones	El cocinero debe haber iniciado sesión y existir pedidos pendientes.
Datos de Prueba	Pedido N°: 105 Producto: Arroz con pollo Estado inicial: Pendiente
Pasos a Ejecutar	1. Ingresar al sistema como cocinero. 2. Visualizar los pedidos pendientes. 3. Seleccionar un pedido. 4. Presionar la opción “Marcar en preparación”.
Resultado Esperado	El estado del pedido cambia a “En preparación”.
Resultado Obtenido	El sistema actualiza el estado del pedido a “En preparación”.

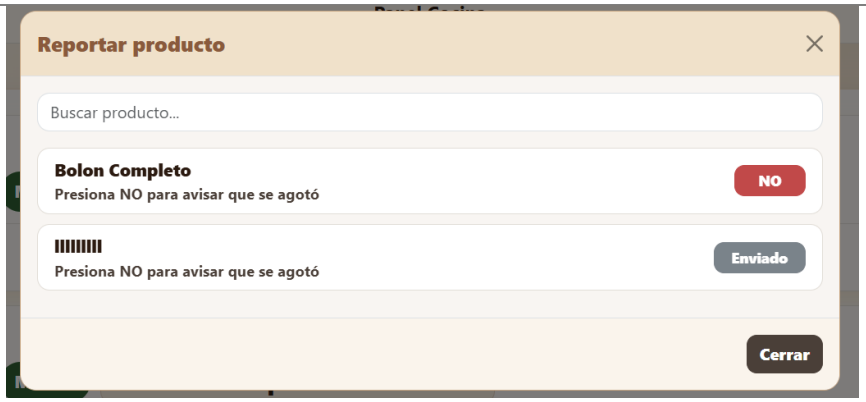
Evidencia	 

Caso de Prueba – HU22 Marcar pedido como listo	
Código del Caso de Prueba	CP-29
Historia de Usuario	HU22 – Marcar pedido como listo
Nombre del Caso de Prueba	Marcar pedido como listo
Objetivo	Validar que el cocinero pueda marcar un pedido como listo para ser entregado.
Usuario	Cocinero

Precondiciones	El cocinero debe haber iniciado sesión y existir pedidos en preparación.
Datos de Prueba	Pedido N°: 105 Producto: Arroz con pollo Estado inicial: En preparación
Pasos a Ejecutar	1. Ingresar al sistema como cocinero. 2. Acceder al módulo de pedidos en preparación. 3. Seleccionar un pedido. 4. Presionar la opción “Marcar como listo”.
Resultado Esperado	El estado del pedido cambia a “Listo”.
Resultado Obtenido	El sistema actualiza el estado del pedido a “Listo” y notifica al mesero o cajero.
Evidencia	

Caso de Prueba – HU23 Avisar producto agotado	
Código del Caso de Prueba	CP-30
Historia de Usuario	HU23 – Avisar producto agotado

Nombre del Caso de Prueba	Avisar producto agotado
Objetivo	Validar que el cocinero pueda registrar un producto como agotado para evitar que siga siendo ofrecido.
Usuario	Cocinero
Precondiciones	El cocinero debe haber iniciado sesión.
Datos de Prueba	Pedido N°: 105 Producto: Arroz con pollo Estado inicial: Disponible
Pasos a Ejecutar	1. Ingresar al sistema como cocinero. 2. Acceder al módulo de productos. 3. Seleccionar el producto agotado. 4. Marcar la opción “Producto agotado”. 5. Guardar cambios.
Resultado Esperado	El producto queda registrado como agotado y no aparece disponible para nuevos pedidos.
Resultado Obtenido	El sistema marca el producto como agotado y deja de mostrarse para nuevos pedidos.

Evidencia	  

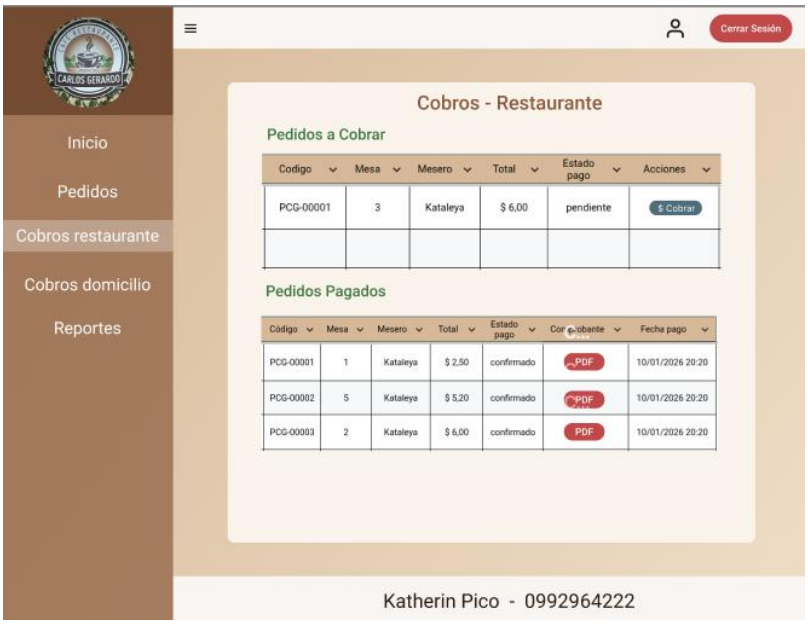
ANEXO H: ITERACIÓN 7 – GESTIÓN DE PAGOS Y COBROS

PLANIFICACIÓN

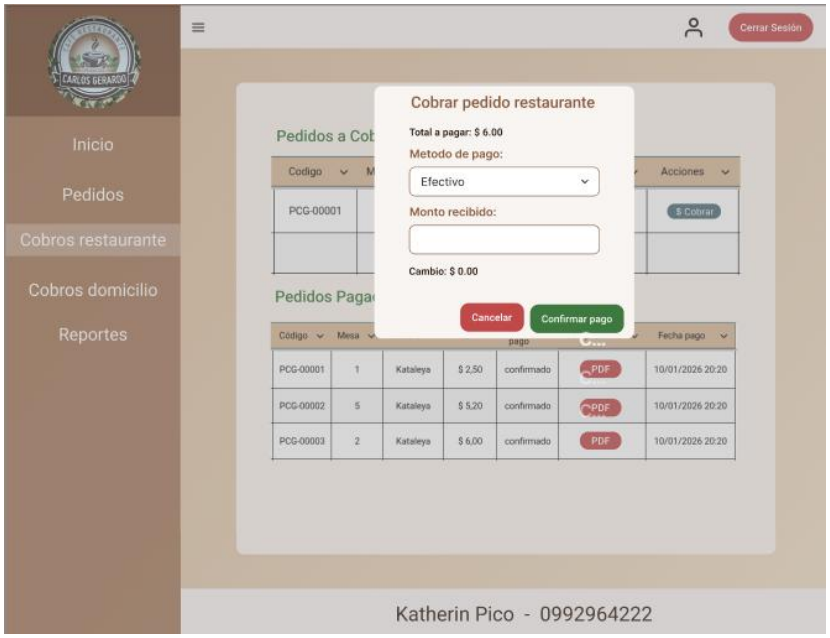
Número	Historia de Usuario
HU27	Registrar pago del pedido del restaurante
HU28	Ver total del pedido
HU29	Aplicar recargo por servicio
HU30	Generar comprobante de pago
HU41	Registrar pago de pedidos a domicilio

DISEÑO

Se observa el apartado para realizar cobros de los pedidos dentro del restaurante, en donde se visualiza una lista con los datos del pedido con su total a cobrar y una lista de los pedidos ya pagados con su respectivo comprobante en PDF.



Se observa un pequeño SweetAlert en el cual se podrá registrar el pago de un pedido, en el cual se muestra el total a cobrar del pedido, el método de pago por el cual se va a cobrar, el monto recibido y, en caso de haber cambio, también se mostrará.



Se visualiza la nota de venta del pedido dentro del restaurante, el cual ya fue cobrado.

Café Restaurante "Productos Carlos Gerardo"

Telf: 0992964222
Dirección: Barrio Eloy Alfaro

NOTA DE VENTA

Nº: C-223-94
Fecha: 14/01/2026 21:20
Tipo: Restaurante
Mesa: 13
Cliente: Consumidor final

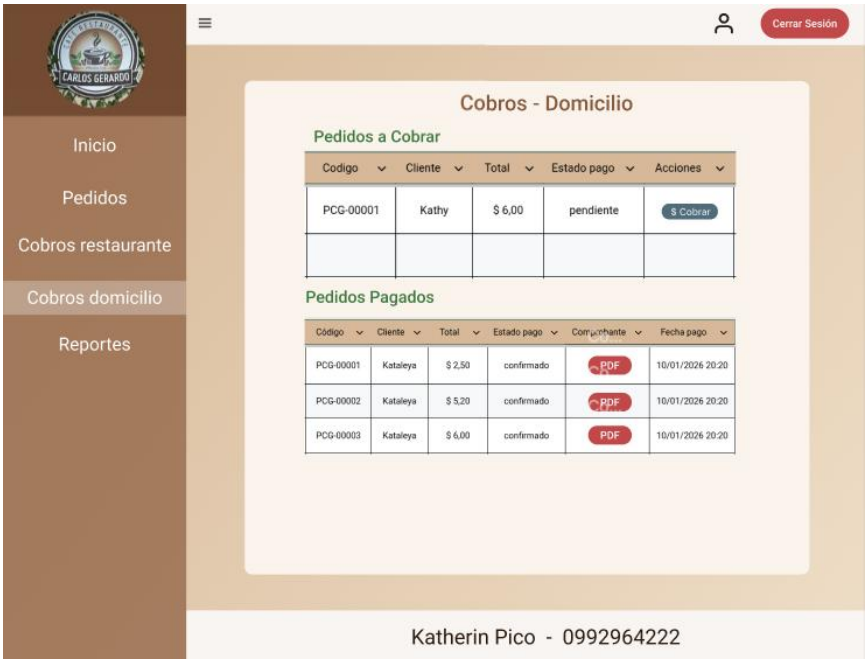
Descripción	Cant.	Sub.	P.Total
hhhhhhhhhh	1	$ 3,50	$ 3,50

TOTAL GENERAL: $ 3,50

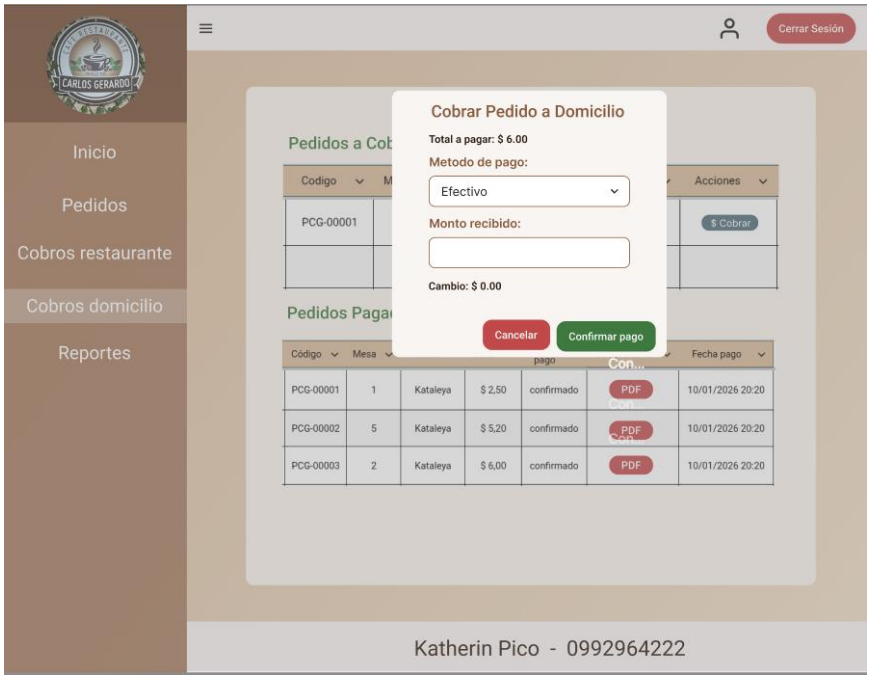
Método de pago: efectivo
Recibido: $ 5,00
Cambio: $ 1,50

Documento NO VÁLIDO para efectos tributarios.

Se observa el apartado para realizar cobros de los pedidos a domicilio, en donde se visualiza una lista con los datos del pedido con su total a cobrar y una lista de los pedidos ya pagados con su respectivo comprobante en PDF.



Se observa un pequeño SweetAlert en el cual se podrá registrar el pago de un pedido, en el cual se muestra el total a cobrar del pedido, el método de pago por el cual se va a cobrar, el monto recibido y, en caso de haber cambio, también se mostrará.



Se visualiza la nota de venta del pedido a domicilio, el cual ya fue cobrado.

Café Restaurante "Productos Carlos Gerardo"

Telf: 0992964222

Dirección: Barrio Eloy Alfaro

NOTA DE VENTA

N°: D-227-96
Fecha: 14/01/2026 21:46
Cliente: Consumidor final

Descripción	Cant.	Sub.	Recargo U.	P.Total
hhhhh	1	$ 2,00	$ 0,20	$ 2,20

TOTAL GENERAL: $ 2,20

Método de pago: efectivo
Recibido: $ 5,00
Cambio: $ 2,80

Documento NO VÁLIDO para efectos tributarios.

CODIFICACIÓN

Se evidencia el código del template en donde se va a mostrar una lista de los pedidos a cobrar y ya pagados registrados en la base de datos.

```
<div class="modal fade" id="modalCobro" tabindex="-1">
    <div class="modal-dialog">
        <div class="modal-content shadow-lg">

            <div class="modal-header bg-primary text-white">
                <h5 class="modal-title">Cobrar Pedido Restaurante</h5>
                <button class="btn-close" data-bs-dismiss="modal"></button>
            </div>

            <div class="modal-body">
                <input type="hidden" id="pedido_id">

                <p><strong>Total a pagar:</strong> $ <span id="total_pagar"></span></p>

                <!-- DATOS CLIENTE (solo si total >= 50) -->
                <div id="contenedor_cliente_factura" class="mt-3" style="display:none;">
                <label class="form-label">Nombre del cliente</label>
                <input type="text" class="form-control" id="cliente_nombre_factura" placeholder="Nombre y apellido">

                <label class="form-label mt-2">Dirección del cliente</label>
                <input type="text" class="form-control" id="cliente_direccion_factura" placeholder="Dirección">
                </div>


                <label class="form-label">Método de pago</label>
                <select class="form-select" id="metodo_pago">
                    <option value="efectivo">Efectivo</option>
                    <option value="transferencia">Transferencia</option>
                </select>

                <div id="contenedor_monto" class="mt-3">
                    <label class="form-label">Monto recibido</label>
                    <input type="number" class="form-control" id="monto_recibido" min="0" step="0.01">
                </div>

                <div id="contenedor_referencia" class="mt-3" style="display:none;">
                    <label class="form-label">Número de comprobante</label>
                    <input type="text" class="form-control" id="referencia_transferencia">
                </div>

                <div id="contenedor_monto_transfer" class="mt-3" style="display:none;">
                    <label class="form-label">Monto transferido</label>
                    <input type="number" class="form-control" id="monto_transferencia" min="0" step="0.01">
                </div>
```

Se evidencia el código del models (modelo), en donde se está creando la tabla de pagos para el registro en la base de datos.

```python
# ----------------------------
# Pago
# ----------------------------
class Pago(models.Model):
    METODOS = [
        ('efectivo', 'Efectivo'),
        ('transferencia', 'Transferencia'),
    ]
    ESTADOS = [
        ('pendiente', 'Pendiente'),
        ('confirmado', 'Confirmado'),
    ]

    pedido = models.ForeignKey(Pedido, on_delete=models.CASCADE, related_name='pagos')
    total = models.DecimalField(max_digits=10, decimal_places=2)
    metodo_pago = models.CharField(max_length=20, choices=METODOS)
    monto_recibido = models.DecimalField(max_digits=10, decimal_places=2, default=0)
    cambio = models.DecimalField(max_digits=10, decimal_places=2, default=0)
    referencia_transferencia = models.CharField(max_length=100, blank=True, null=True)
    estado_pago = models.CharField(max_length=20, choices=ESTADOS, default='pendiente')
    fecha_hora = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return f"Pago Pedido {self.pedido.id} - {self.total}"
```

Se observa el código del views en donde se programa la lógica del backend para realizar los cobros de los pedidos registrados por el mesero y el cajero.

```python
#------------------------------
# CAJERO - COBROS
#------------------------------
@login_required(login_url='login')
@rol_requerido('cajero')
def cajero_restaurante_cobros(request):

    pagos_confirmados = Pago.objects.filter(
        pedido=OuterRef('pk'),
        estado_pago='confirmado'
    )

    hoy = timezone.localdate()

    # ---------- PEDIDOS PENDIENTES DE COBRO (SOLO HOY) ----------
    pedidos_cobrar = (
        Pedido.objects
        .filter(
            tipo_pedido='restaurante',
            estado__in=['listo', 'finalizado'],
            fecha_hora__date=hoy
        )
        .annotate(tiene_pago=Exists(pagos_confirmados))
        .filter(tiene_pago=False)
        .order_by('id')
    )

    # ---------- PEDIDOS PAGADOS (SOLO HOY) + COMPROBANTES ----------
    pedidos_pagados = (
        Pedido.objects
        .filter(
            tipo_pedido='restaurante',
            pagos__estado_pago='confirmado',
            pagos__fecha_hora__date=hoy
        )
        .prefetch_related('pagos__comprobante_set')  # para acceder a los PDFs
        .distinct()
        .order_by('id')
    )

    # añadir el último pago confirmado a cada pedido (igual que domicilio)
    for p in pedidos_pagados:
        p.pago_confirmado = (
            p.pagos
             .filter(estado_pago='confirmado')
```

Se evidencia el código del template en donde se va a mostrar la nota de venta de los cobros registrados en la base de datos.

```
    <p class="subtitulo">NOTA DE VENTA</p>

    <div class="info">
        <b>N°:</b> {{ comprobante.numero_comprobante }} <br>
        <b>Fecha:</b> {{ comprobante.fecha_hora|date:"d/m/Y H:i" }} <br>
        <b>Cliente:</b>
        {% if requiere_datos %}
            {{ nombre_impreso|default:"(SIN NOMBRE)" }} <br>
            <b>Dirección:</b> {{ direccion_impresa|default:"(SIN DIRECCIÓN)" }} <br>
        {% else %}
            Consumidor final <br>
        {% endif %}

    </div>

    <hr>

    <table>
        <thead>
            <tr>
                <th>Descripción</th>
                <th>Cant.</th>
                <th>Sub.</th>
                <th>Recargo U.</th>
                <th>P.Total</th>
            </tr>
        </thead>

        <tbody>
        {% for d in detalles %}
            <tr>
                <td>{{ d.producto.nombre }}</td>
                <td class="center">{{ d.cantidad }}</td>
                <td class="right">$ {{ d.precio_unitario|floatformat:2 }}</td>
                <td class="right">$ {{ d.recargo|floatformat:2 }}</td>
                <td class="right">$ {{ d.total_con_recargo|floatformat:2 }}</td>
            </tr>
        {% endfor %}
        </tbody>
    </table>
```

Se evidencia el código del models (modelo), en donde se está creando la tabla de comprobante para el registro en la base de datos.

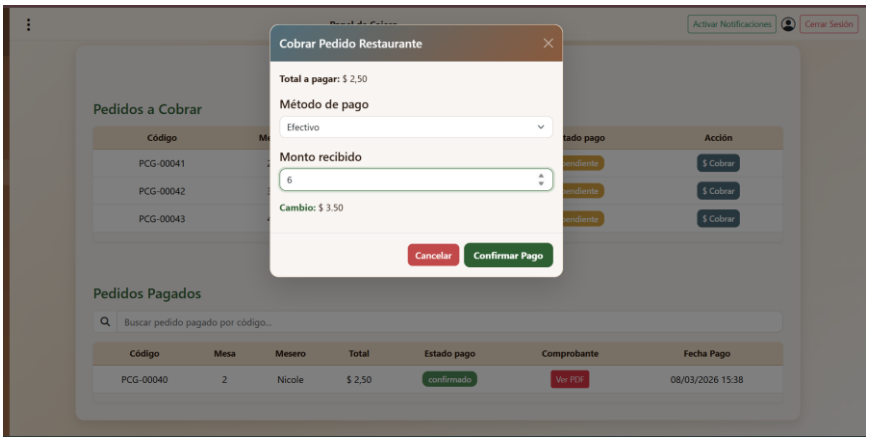
```
# ----------------------------
# Comprobante
# ----------------------------
class Comprobante(models.Model):
    pago = models.ForeignKey(Pago, on_delete=models.CASCADE)
    numero_comprobante = models.CharField(max_length=50, unique=True)
    fecha_hora = models.DateTimeField(auto_now_add=True)
    nombre_cliente = models.CharField(max_length=100, blank=True, null=True)
    direccion_cliente = models.CharField(max_length=255, blank=True, null=True)
    correo_cliente = models.EmailField(blank=True, null=True)

    archivo_pdf = models.FileField(
        upload_to='comprobantes/',
        blank=True,
        null=True
    )

    def __str__(self):
        return f"Comprobante {self.numero_comprobante}"
```

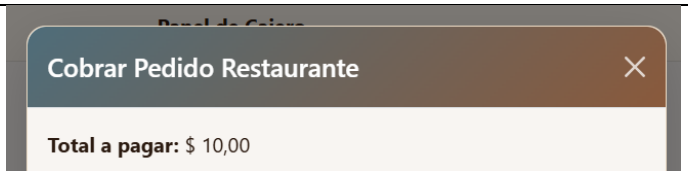
Se observa el código del views en donde se programa la lógica del backend para generar el comprobante en un archivo PDF registrado por el cajero.

```python
def generar_comprobante_pdf(comprobante):
    pago = comprobante.pago
    pedido = pago.pedido
    detalles = pedido.detalles.all()

    requiere_datos = pago.total >= Decimal("50.00")

    # USA LO GUARDADO EN EL COMPROBANTE (NO el pedido)
    nombre_impreso = (comprobante.nombre_cliente or "").strip()
    direccion_impresa = (comprobante.direccion_cliente or "").strip()

    if requiere_datos:
        if not nombre_impreso or not direccion_impresa:
            raise ValueError("Pedido >= $50 requiere nombre y dirección del cliente.")
    else:
        nombre_impreso = "Consumidor final"
        direccion_impresa = ""

    # (opcional) asegúrate de persistir lo final
    comprobante.nombre_cliente = nombre_impreso
    comprobante.direccion_cliente = direccion_impresa
    comprobante.save(update_fields=["nombre_cliente", "direccion_cliente"])

    template = "cajero/comprobantes/comprobante_restaurante.html" if pedido.tipo_pedido == "restaurante" else "cajero/comprobantes/comprobante_domicilio.html"

    html_string = render_to_string(template, {
        "comprobante": comprobante,
        "pago": pago,
        "pedido": pedido,
        "detalles": detalles,
        "requiere_datos": requiere_datos,
        "nombre_impreso": nombre_impreso,
        "direccion_impresa": direccion_impresa,
    })

    pdf_file = HTML(string=html_string).write_pdf()
    nombre_archivo = f"{comprobante.numero_comprobante}.pdf"
    comprobante.archivo_pdf.save(nombre_archivo, ContentFile(pdf_file), save=True)
```

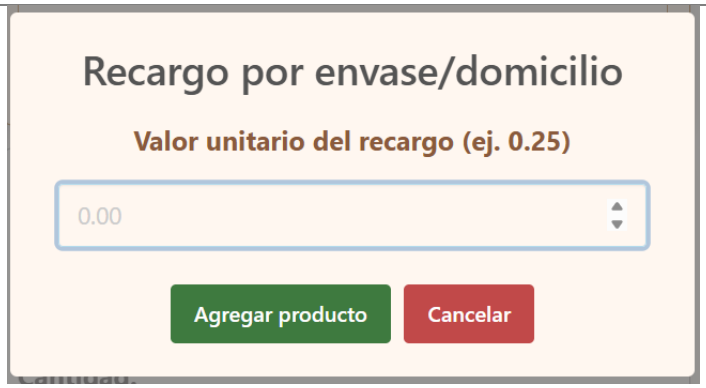
PRUEBAS

Caso de Prueba – HU27 Registrar pago del pedido del restaurante	
Código del Caso de Prueba	CP-31
Historia de Usuario	HU27 – Registrar pago del pedido del restaurante
Nombre del Caso de Prueba	Registrar pago del pedido del restaurante
Objetivo	Validar que el cajero pueda registrar el pago de un pedido realizado en el restaurante.
Usuario	Cajero
Precondiciones	El cajero debe haber iniciado sesión y existir un pedido registrado pendiente de pago.
Datos de Prueba	Pedido N°: 110 Total del pedido: $12.50 Método de pago: Efectivo

Pasos a Ejecutar	1. Ingresar al sistema como cajero. 2. Acceder al módulo de pedidos del restaurante. 3. Seleccionar un pedido pendiente de pago. 4. Registrar el método de pago. 5. Confirmar el pago.
Resultado Esperado	El sistema registra correctamente el pago del pedido y actualiza su estado a pagado.
Resultado Obtenido	El sistema registra el pago del pedido y lo marca como pagado.
Evidencia	 

Caso de Prueba – HU28 Ver total del pedido	
Código del Caso de Prueba	CP-32
Historia de Usuario	HU28 – Ver total del pedido
Nombre del Caso de Prueba	Ver total del pedido
Objetivo	Validar que el sistema calcule y muestre automáticamente el total del pedido.
Usuario	Cajero
Precondiciones	El cajero debe haber iniciado sesión y existir un pedido con productos agregados.
Datos de Prueba	Producto 1: Arroz con pollo – $5.00 Producto 2: Jugo natural – $2.50 Cantidad: 2
Pasos a Ejecutar	1. Ingresar al sistema como cajero. 2. Seleccionar un pedido registrado. 3. Visualizar los productos agregados al pedido. 4. Revisar el total calculado por el sistema.
Resultado Esperado	El sistema calcula automáticamente el total del pedido sumando los productos.
Resultado Obtenido	El sistema muestra correctamente el total calculado del pedido.
Evidencia	

Caso de Prueba – HU29 Aplicar recargo por servicio	
Código del Caso de Prueba	CP-33
Historia de Usuario	HU29 – Aplicar recargo por servicio
Nombre del Caso de Prueba	Aplicar recargo por servicio
Objetivo	Validar que el sistema aplique un recargo cuando el pedido es a domicilio.
Usuario	Cajero
Precondiciones	El cajero debe haber iniciado sesión y existir un pedido a domicilio registrado.
Datos de Prueba	Pedido N°: 111 Total del pedido: $10.00 Recargo por domicilio: $1.00
Pasos a Ejecutar	1. Ingresar al sistema como cajero. 2. Acceder al módulo de pedidos a domicilio. 3. Seleccionar un pedido registrado. 4. Verificar el cálculo del recargo por servicio.
Resultado Esperado	El sistema aplica automáticamente el recargo al total del pedido.
Resultado Obtenido	El sistema suma el recargo por servicio al total del pedido.

Evidencia	

Caso de Prueba – HU30 Generar comprobante de pago	
Código del Caso de Prueba	CP-34
Historia de Usuario	HU30 – Generar comprobante de pago
Nombre del Caso de Prueba	Generar comprobante de pago
Objetivo	Validar que el sistema genere un comprobante de pago luego de registrar el pago del pedido.
Usuario	Cajero
Precondiciones	El cajero debe haber iniciado sesión y existir un pedido pagado.
Datos de Prueba	Pedido N°: 112 Total pagado: $15.00 Cliente: Juan Pérez
Pasos a Ejecutar	1. Ingresar al sistema como cajero. 2. Seleccionar un pedido pagado. 3. Presionar la opción “Generar comprobante”. 4. Visualizar o descargar el comprobante.

Resultado Esperado	El sistema genera correctamente el comprobante de pago del pedido.
Resultado Obtenido	El sistema genera el comprobante de pago con los datos del pedido.
Evidencia	

Café Restaurante "Productos Carlos Gerardo"
Telf: 0992964222
Dirección: Barrio Eloy Alfaro

NOTA DE VENTA

Nº: C-223-94
Fecha: 14/01/2026 21:20
Tipo: Restaurante
Mesa: 13
Cliente: Consumidor final

Descripción	Cant.	Sub.	P.Total
hhhhhhhhhh	1	$ 3,50	$ 3,50

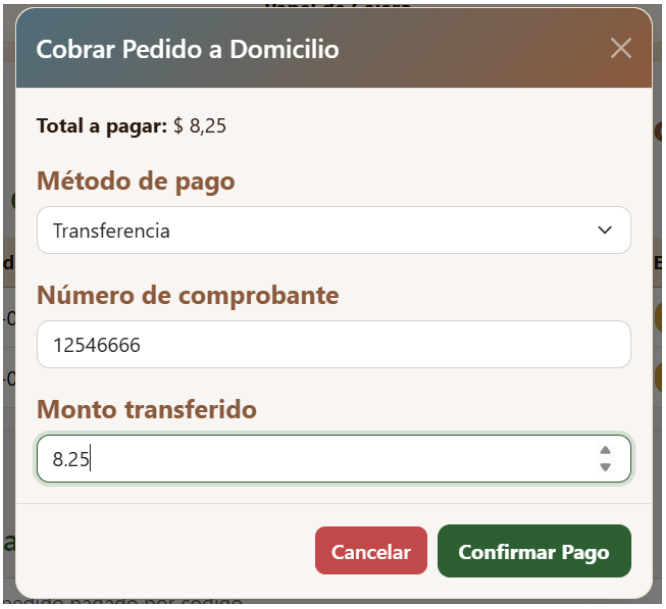
TOTAL GENERAL: $ 3,50

Método de pago: efectivo
Recibido: $ 5,00
Cambio: $ 1,50

Documento NO VÁLIDO para efectos tributarios.

Caso de Prueba – HU41 Registrar pago de pedidos a domicilio	
Código del Caso de Prueba	CP-35
Historia de Usuario	HU41 – Registrar pago de pedidos a domicilio
Nombre del Caso de Prueba	Registrar pago de pedidos a domicilio
Objetivo	Validar que el cajero pueda registrar el pago de pedidos realizados a domicilio.
Usuario	Cajero

Precondiciones	El cajero debe haber iniciado sesión y existir un pedido a domicilio pendiente de pago.
Datos de Prueba	Pedido N°: 113 Cliente: Ana Torres Total del pedido: $14.00 Método de pago: Transferencia
Pasos a Ejecutar	1. Ingresar al sistema como cajero. 2. Acceder al módulo de pedidos a domicilio. 3. Seleccionar un pedido pendiente de pago. 4. Registrar el método de pago. 5. Confirmar el pago.
Resultado Esperado	El sistema registra el pago del pedido a domicilio y actualiza su estado a pagado.
Resultado Obtenido	El sistema registra el pago del pedido a domicilio y genera el comprobante correspondiente.

Evidencia	 

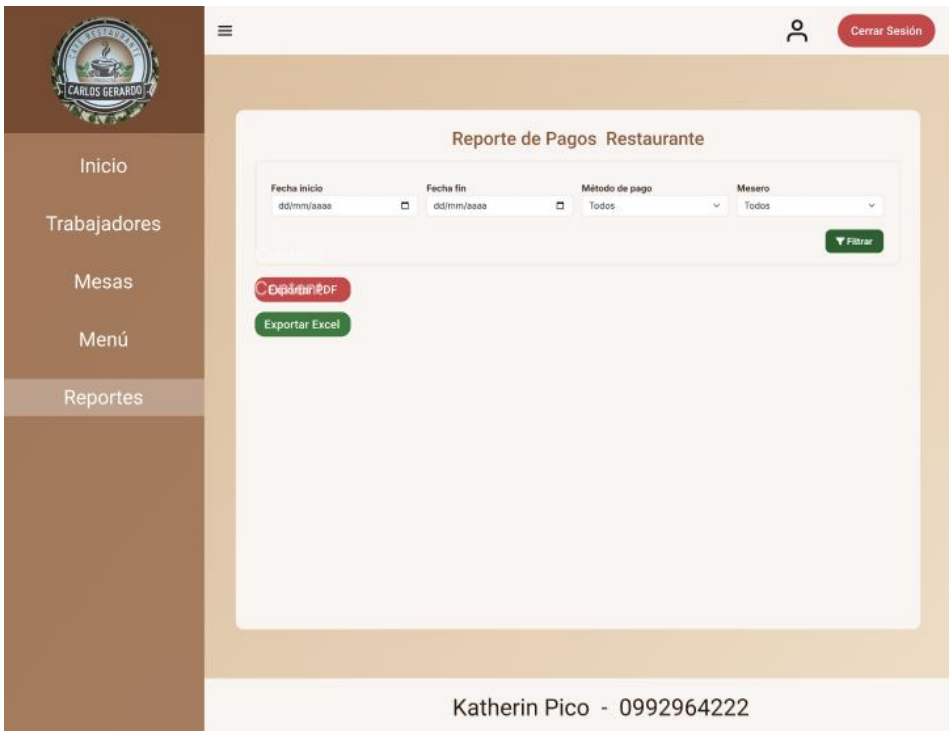
ANEXO I: ITERACIÓN 8 – REPORTES Y CONTROL ADMINISTRATIVO

PLANIFICACIÓN

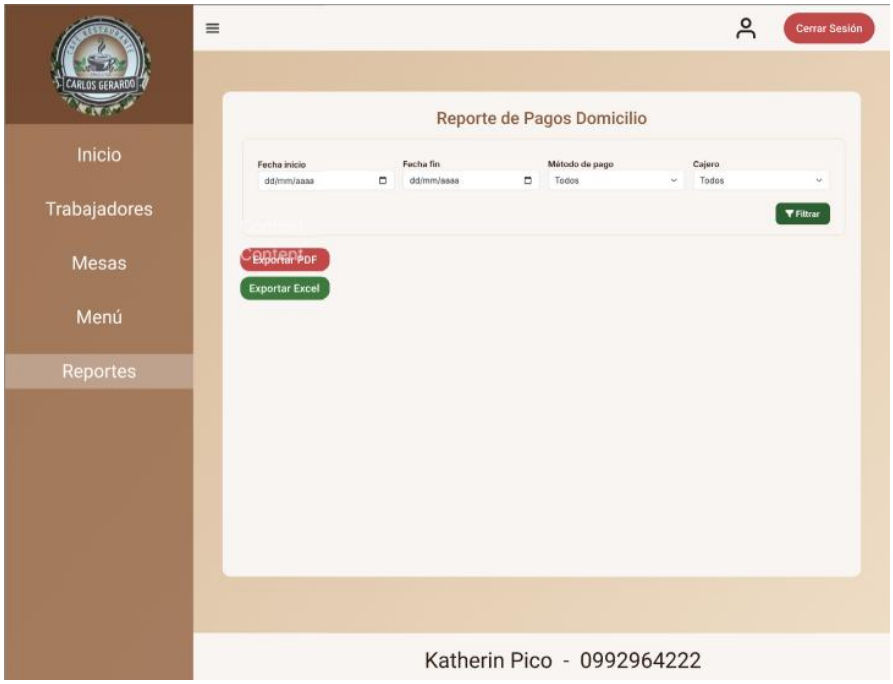
Número	Historia de Usuario
HU07	Visualizar reportes de ventas
HU10	Visualizar reportes por tipo de pedido
HU11	Visualizar reporte unificado de cobros
HU42	Visualizar reporte unificado para cierre de caja

DISEÑO

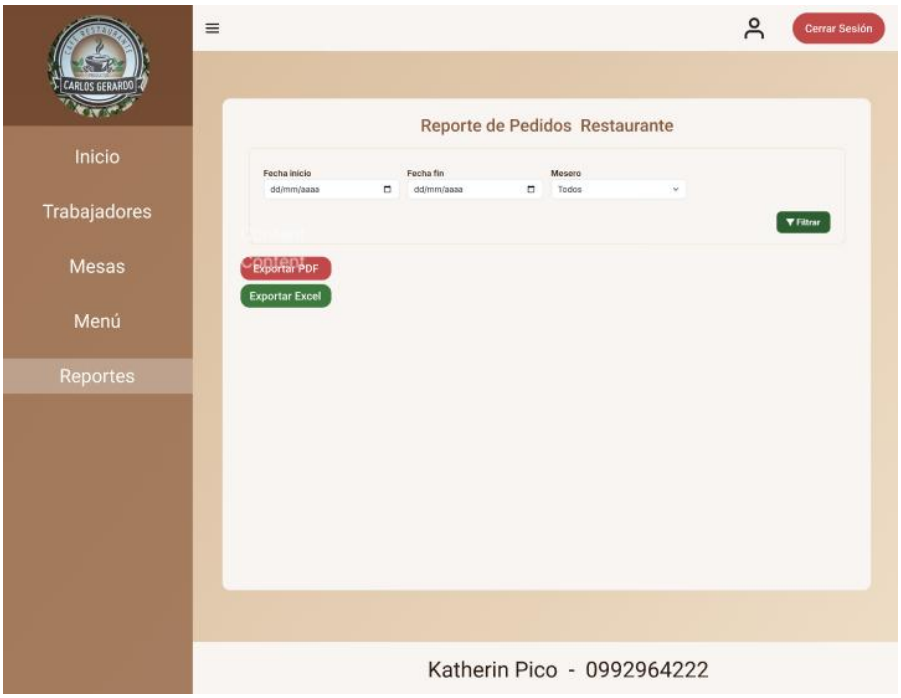
Se observa el reporte de pagos de pedidos dentro del restaurante, el cual tiene un filtro por fecha de inicio, fecha de fin, método de pago y el mesero encargado de ese pedido; así mismo tiene las opciones de exportar en PDF y Excel.



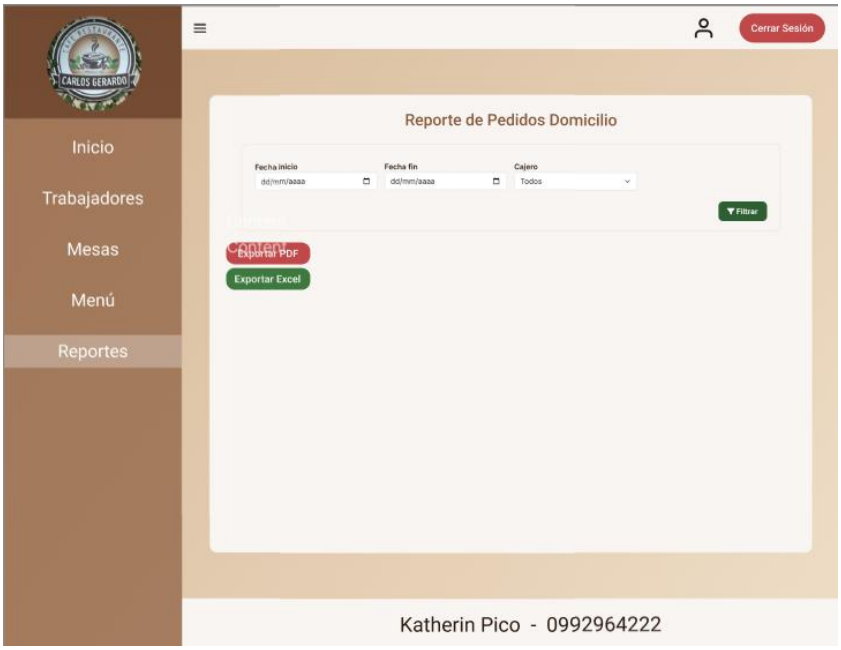
Se observa el reporte de pagos de pedidos a domicilio, el cual tiene un filtro por fecha de inicio, fecha de fin, método de pago y cajero encargado de ese pedido; además, cuenta con la función de exportar en PDF y Excel.



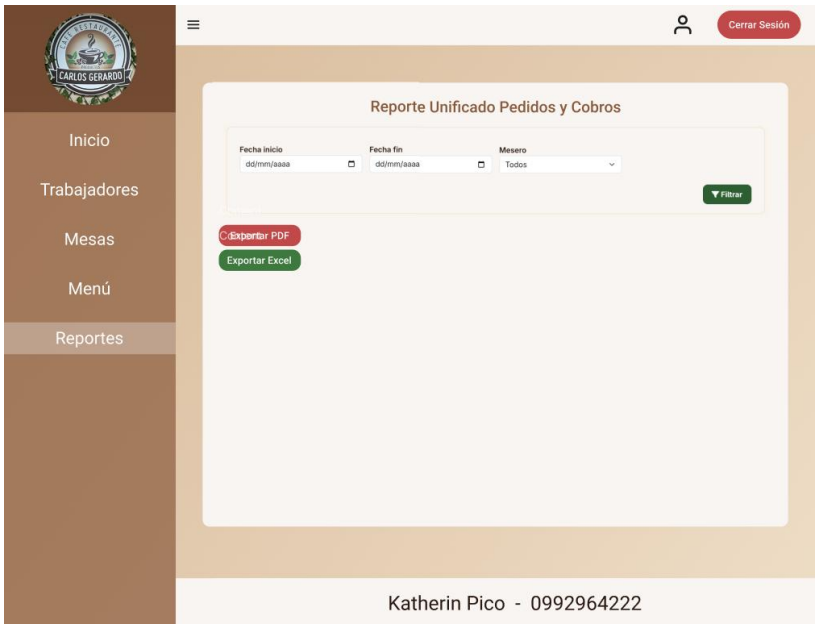
Se observa el reporte de pedidos dentro del restaurante, el cual tiene un filtro por fecha de inicio, fecha de fin y por el mesero encargado de ese pedido; además, cuenta con la opción de exportar en PDF y Excel.



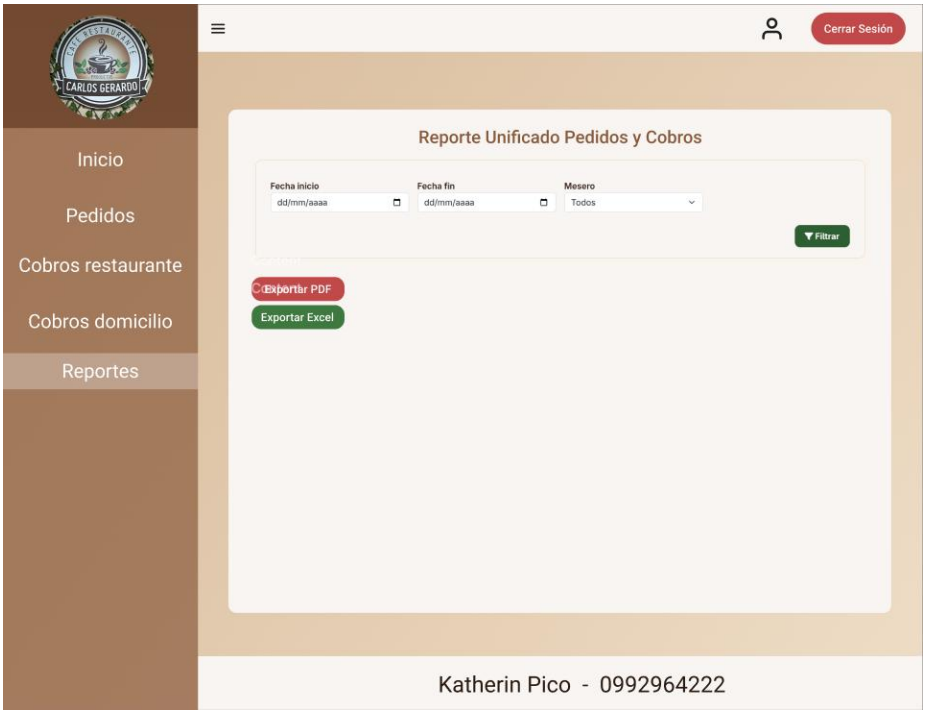
Se observa el reporte de pedidos a domicilio, el cual tiene un filtro por fecha de inicio, fecha de fin y por el cajero encargado de ese pedido; además, cuenta con la opción de exportar en PDF y Excel.



Se observa el reporte de pagos de pedidos dentro del restaurante y a domicilio, el cual tiene un filtro por fecha de inicio, fecha de fin, método de pago y personal encargado de ese pedido; además, cuenta con la función de exportar en PDF y Excel.



Se observa el reporte de pagos de pedidos dentro del restaurante y a domicilio, el cual tiene un filtro por fecha de inicio, fecha de fin, método de pago y personal encargado de ese pedido; además, cuenta con la función de exportar en PDF y Excel, la cual permitirá al cajero realizar cierre de caja.



CODIFICACIÓN

Se observa el código del views en donde se programa la lógica del backend para generar reportes de pedidos del restaurante.

```python
#----------------------------
# Reportes
#----------------------------
def reportes_general(request):
    return render(request, 'administrador/reportes/reportes_general.html')

@login_required(login_url='login')
@rol_requerido('admin')
def reporte_pagos_restaurante(request):

    pagos = Pago.objects.filter(
        pedido__tipo_pedido='restaurante',
        estado_pago='confirmado'
    ).select_related('pedido')

    fecha_inicio = request.GET.get("inicio")
    fecha_fin = request.GET.get("fin")
    metodo = request.GET.get("metodo")
    mesero_id = request.GET.get("mesero")

    if fecha_inicio:
        pagos = pagos.filter(fecha_hora__date__gte=fecha_inicio)

    if fecha_fin:
        pagos = pagos.filter(fecha_hora__date__lte=fecha_fin)

    if metodo and metodo != "todos":
        pagos = pagos.filter(metodo_pago=metodo)

    if mesero_id and mesero_id != "todos":
        pagos = pagos.filter(pedido__mesero_id=mesero_id)

    total_recaudado = pagos.aggregate(total=Sum('total'))['total'] or 0

    meseros = Usuario.objects.filter(rol='mesero')

    return render(request, "administrador/reportes/reporte_restaurante.html", {
        "pagos": pagos,
        "total_recaudado": total_recaudado,
        "meseros": meseros,
    })

@login_required(login_url='login')
@rol_requerido('admin')
```

Se observa el código del views en donde se programa la lógica del backend para generar reportes de pedidos a domicilio

```python
@login_required(login_url='login')
@rol_requerido('admin')
def reporte_pedidos_domicilio(request):

    pedidos = Pedido.objects.filter(
        tipo_pedido='domicilio'
    ).select_related('cajero')

    # Filtros
    fecha_inicio = request.GET.get("inicio")
    fecha_fin = request.GET.get("fin")
    estado = request.GET.get("estado")
    cajero_id = request.GET.get("cajero")

    if fecha_inicio:
        pedidos = pedidos.filter(fecha_hora__date__gte=fecha_inicio)

    if fecha_fin:
        pedidos = pedidos.filter(fecha_hora__date__lte=fecha_fin)

    if estado and estado != "todos":
        pedidos = pedidos.filter(estado=estado)

    if cajero_id and cajero_id != "todos":
        pedidos = pedidos.filter(cajero_id=cajero_id)

    total_pedidos = pedidos.count()
    total_ventas = pedidos.aggregate(total=Sum('total'))['total'] or 0

    cajeros = Usuario.objects.filter(rol='cajero')

    return render(request, "administrador/reportes/reporte_pedidos_domicilio.html", {
        "pedidos": pedidos,
        "total_pedidos": total_pedidos,
        "total_ventas": total_ventas,
        "cajeros": cajeros,
    })
```

Se evidencia el código del template en donde se va a mostrar los reportes de pedidos dentro del restaurante.

```
204
205         <!-- ================ TOTAL ================ -->
206         <div class="d-flex justify-content-end mb-3">
207             <div class="total-box">
208                 Total recaudado: ${{ total_recaudado }}
209             </div>
210         </div>
211
212         <!-- ================ TABLA ================ -->
213         <div class="table-responsive">
214             <table class="table table-bordered table-hover align-middle">
215                 <thead class="text-center">
216                     <tr>
217                         <th>Código pedido</th>
218                         <th>Cliente</th>
219                         <th>Cajero</th>
220                         <th>Total</th>
221                         <th>Método</th>
222                         <th>Fecha</th>
223                         <th>Comprobante</th>
224                     </tr>
225                 </thead>
226
227                 <tbody>
228                     {% for p in pagos %}
229                     <tr>
230                         <td class="text-center">{{ p.pedido.codigo_pedido }}</td>
231                         <td>{{ p.pedido.nombre_cliente }}</td>
232                         <td>{{ p.pedido.cajero.nombre }}</td>
233                         <td class="text-success fw-bold">${{ p.total }}</td>
234                         <td>{{ p.metodo_pago }}</td>
235                         <td>{{ p.fecha_hora|date:"d/m/Y H:i" }}</td>
236                         <td class="text-center">
237                             {% if p.comprobante_set.first %}
238                             <a href="{{ p.comprobante_set.first.archivo_pdf.url }}" target="_blank"
239                                 class="btn btn-sm btn-danger">
240                                 PDF
241                             </a>
242                             {% else %}
243                             <span class="text-muted">—</span>
244                             {% endif %}
245                         </td>
246
```

Se evidencia el código del template en donde se va a mostrar los reportes de pedidos a domicilio.

```
<!-- ================ FILTROS ================ -->
<form method="GET" class="row g-3 form-filtros mb-4">

    <div class="col-md-3">
        <label class="fw-semibold">Fecha inicio</label>
        <input type="date" name="inicio" class="form-control" value="{{ request.GET.inicio }}">
    </div>

    <div class="col-md-3">
        <label class="fw-semibold">Fecha fin</label>
        <input type="date" name="fin" class="form-control" value="{{ request.GET.fin }}">
    </div>

    <div class="col-md-3">
        <label class="fw-semibold">Tipo de pedido</label>
        <select name="tipo" class="form-select">
            <option value="todos">Todos</option>

            <option value="restaurante" {% if request.GET.tipo == "restaurante" %}selected{% endif %}>
                Restaurante
            </option>

            <option value="domicilio" {% if request.GET.tipo == "domicilio" %}selected{% endif %}>
                Domicilio
            </option>
        </select>
    </div>

    <!-- NUEVO: FILTRO MÉTODO DE PAGO -->
    <div class="col-md-3">
        <label class="fw-semibold">Método de pago</label>
        <select name="metodo" class="form-select">
            <option value="todos">Todos</option>
            <option value="efectivo" {% if request.GET.metodo == "efectivo" %}selected{% endif %}>
                Efectivo
            </option>
            <option value="transferencia" {% if request.GET.metodo == "transferencia" %}selected{% endif %}>
                Transferencia
            </option>
        </select>
    </div>
```

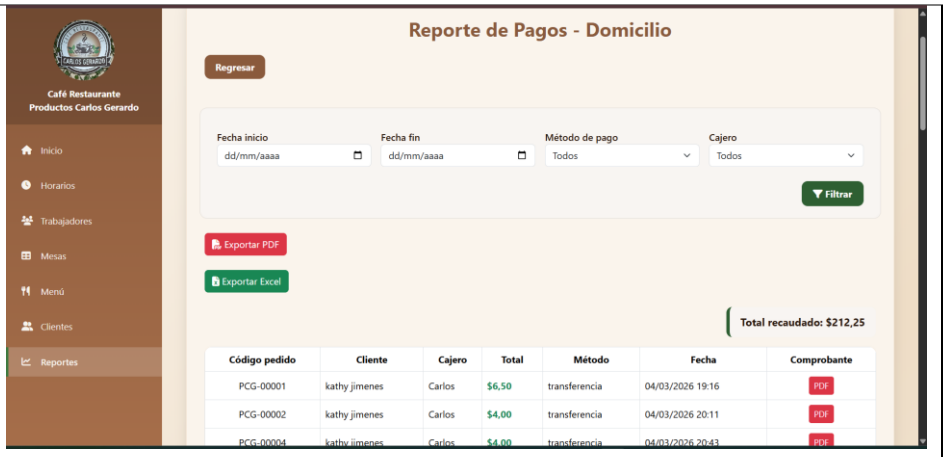
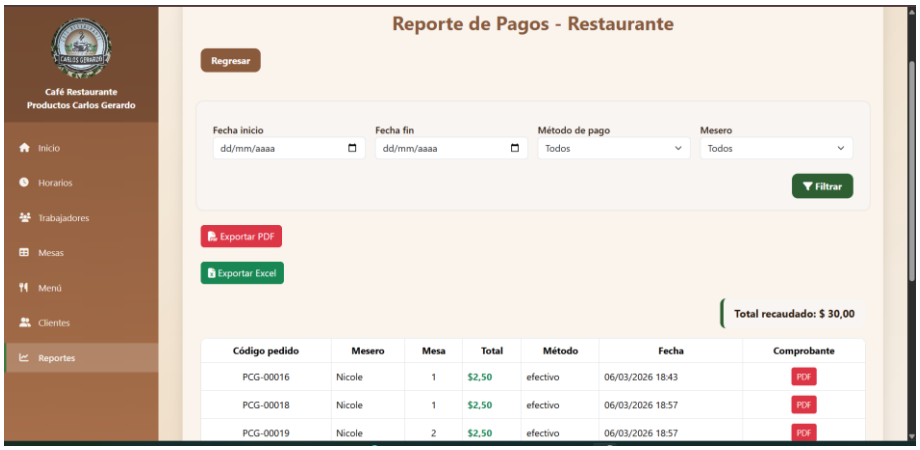
PRUEBAS

Caso de Prueba – HU07 Visualizar reportes de ventas	
Código del Caso de Prueba	CP-36
Historia de Usuario	HU07 – Visualizar reportes de ventas
Nombre del Caso de Prueba	Visualizar reportes de ventas
Objetivo	Validar que el administrador pueda visualizar los reportes de ventas del restaurante.

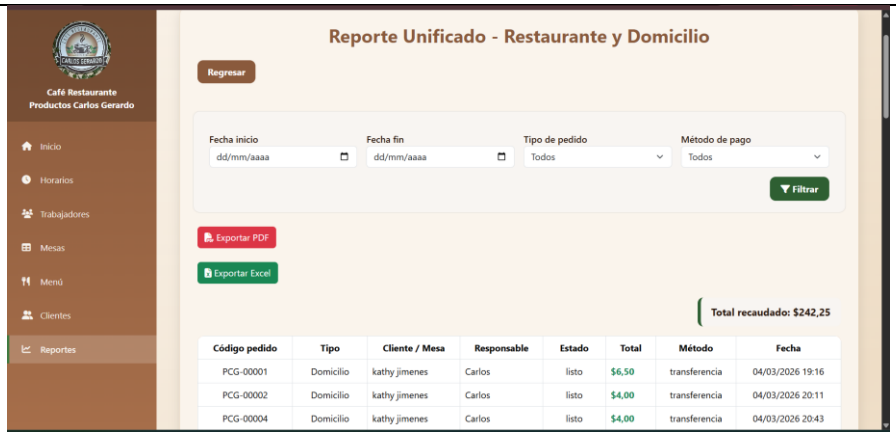
Usuario	Administrador
Precondiciones	El administrador debe haber iniciado sesión en el sistema y existir registros de ventas.
Datos de Prueba	Fecha: 15/05/2026 Ventas registradas: $120.50
Pasos a Ejecutar	1. Ingresar al sistema como administrador. 2. Acceder al módulo de reportes. 3. Seleccionar la opción “Reportes de ventas”. 4. Visualizar la información mostrada.
Resultado Esperado	El sistema muestra correctamente el reporte de ventas registradas.
Resultado Obtenido	El sistema presenta el listado de ventas realizadas en el restaurante.
Evidencia	

Caso de Prueba – HU10 Visualizar reportes por tipo de pedido	
Código del Caso de Prueba	CP-37
Historia de Usuario	HU10 – Visualizar reportes por tipo de pedido

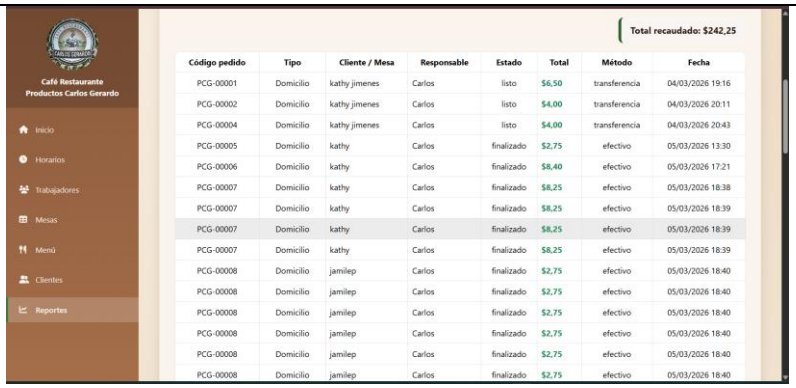
Nombre del Caso de Prueba	Visualizar reportes por tipo de pedido
Objetivo	Validar que el administrador pueda visualizar reportes separados por tipo de pedido (restaurante y domicilio).
Usuario	Administrador
Precondiciones	El administrador debe haber iniciado sesión y existir registros de pedidos en el sistema.
Datos de Prueba	Pedidos restaurante: 8 Pedidos domicilio: 5
Pasos a Ejecutar	1. Ingresar al sistema como administrador. 2. Acceder al módulo de reportes. 3. Seleccionar la opción “Reportes por tipo de pedido”. 4. Revisar los datos mostrados.
Resultado Esperado	El sistema muestra reportes separados para pedidos del restaurante y pedidos a domicilio.
Resultado Obtenido	El sistema presenta correctamente los reportes clasificados por tipo de pedido.

Evidencia	 

Caso de Prueba – HU11 Visualizar reporte unificado de cobros	
Código del Caso de Prueba	CP-38
Historia de Usuario	HU11 – Visualizar reporte unificado de cobros
Nombre del Caso de Prueba	Visualizar reporte unificado de cobros
Objetivo	Validar que el administrador pueda visualizar un reporte general con todos los cobros del restaurante.
Usuario	Administrador

Precondiciones	El administrador debe haber iniciado sesión y existir registros de pagos en el sistema.
Datos de Prueba	Cobros restaurante: $150.00 Cobros domicilio: $75.00
Pasos a Ejecutar	1. Ingresar al sistema como administrador. 2. Acceder al módulo de reportes. 3. Seleccionar la opción “Reporte unificado de cobros”. 4. Visualizar la información mostrada.
Resultado Esperado	El sistema muestra un reporte unificado con todos los cobros realizados.
Resultado Obtenido	El sistema presenta el reporte general de ingresos del restaurante.
Evidencia	

Caso de Prueba – HU42 Visualizar reporte unificado para cierre de caja	
Código del Caso de Prueba	CP-39
Historia de Usuario	HU42 – Visualizar reporte unificado para cierre de caja

Nombre del Caso de Prueba	Visualizar reporte unificado para cierre de caja
Objetivo	Validar que el cajero pueda visualizar el reporte unificado de cobros para realizar el cierre de caja.
Usuario	Cajero
Precondiciones	El cajero debe haber iniciado sesión y existir cobros registrados en el sistema.
Datos de Prueba	Total cobros restaurante: $180.00 Total cobros domicilio: $90.00 Total general: $270.00
Pasos a Ejecutar	1. Ingresar al sistema como cajero. 2. Acceder al módulo de reportes. 3. Seleccionar la opción “Reporte para cierre de caja”. 4. Visualizar los datos mostrados.
Resultado Esperado	El sistema muestra el reporte unificado de cobros necesario para el cierre de caja.
Resultado Obtenido	El sistema presenta correctamente el reporte con el total de cobros del día.
Evidencia	

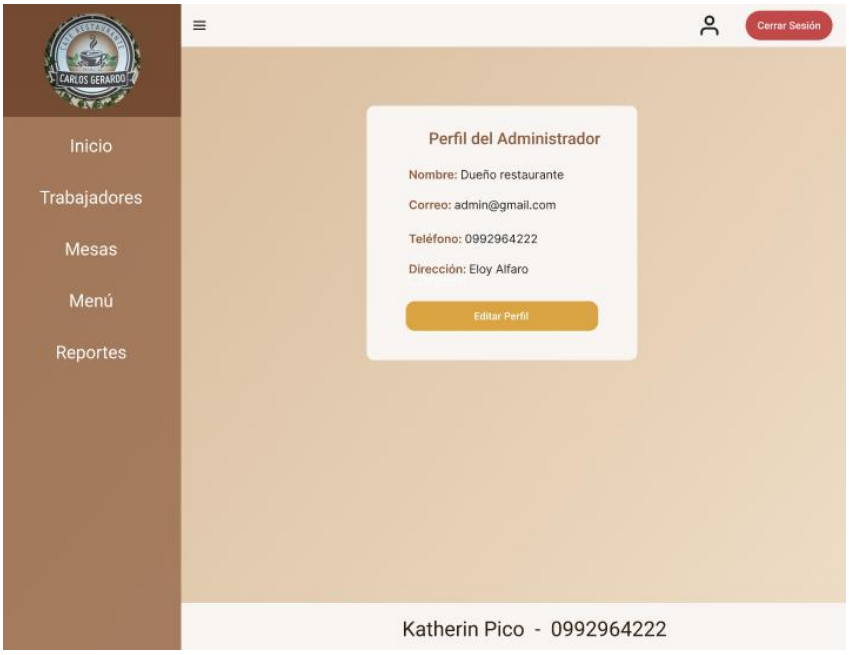
ANEXO J: ITERACIÓN 9 – PERFIL, NOTIFICACIONES Y USO MÓVIL

PLANIFICACIÓN

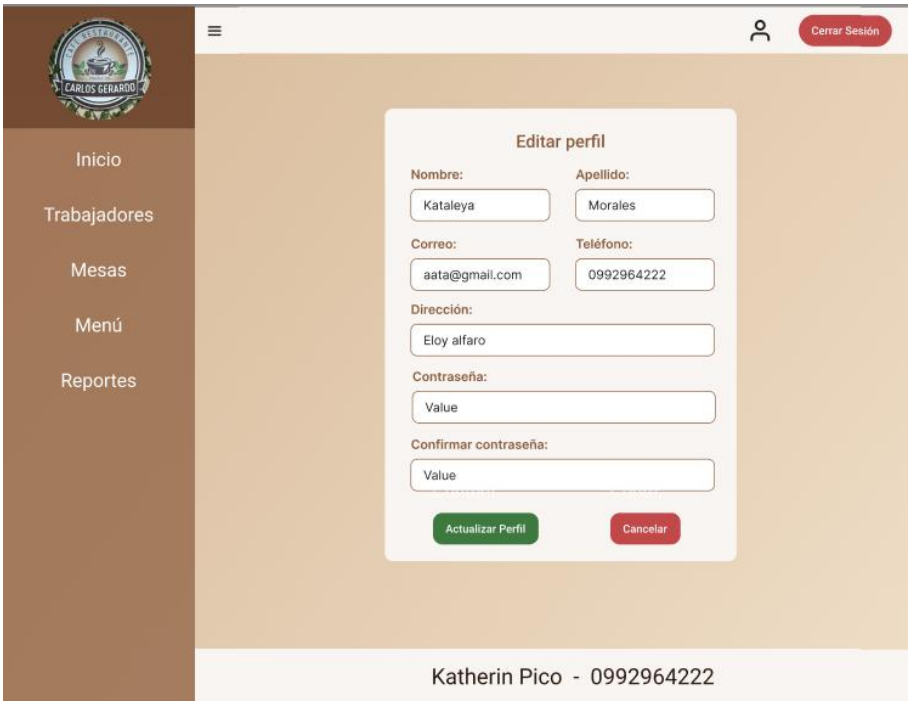
Número	Historia de Usuario
HU12	Editar perfil administrador
HU13	Editar perfil mesero
HU24	Editar perfil cocinero
HU25	Editar perfil cajero
HU33	Recibir notificaciones
HU34	Uso en dispositivos móviles

DISEÑO

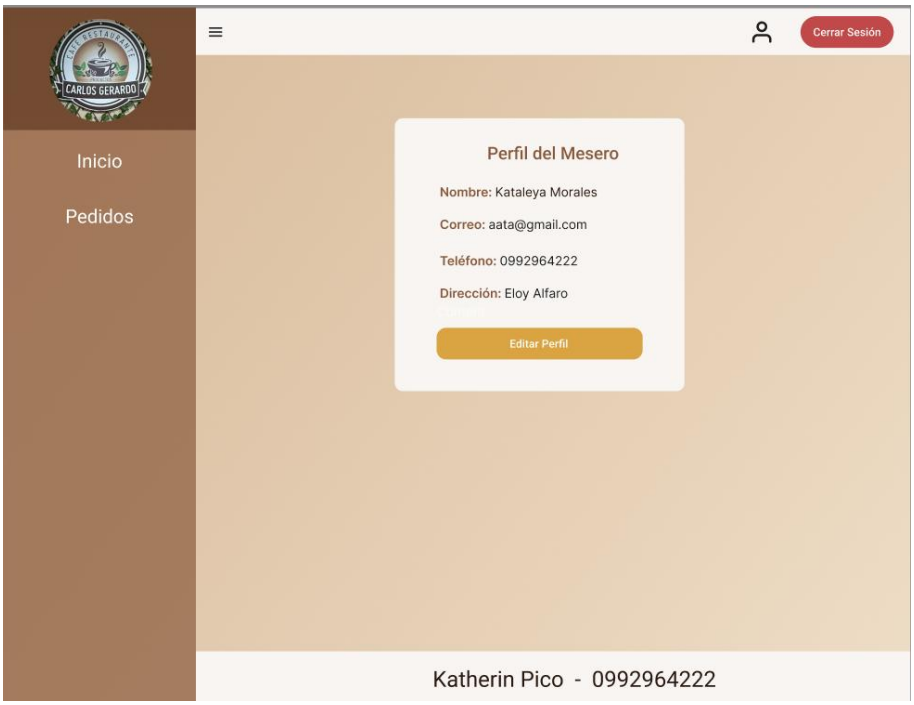
Se observa el perfil del mesero con sus datos personales.



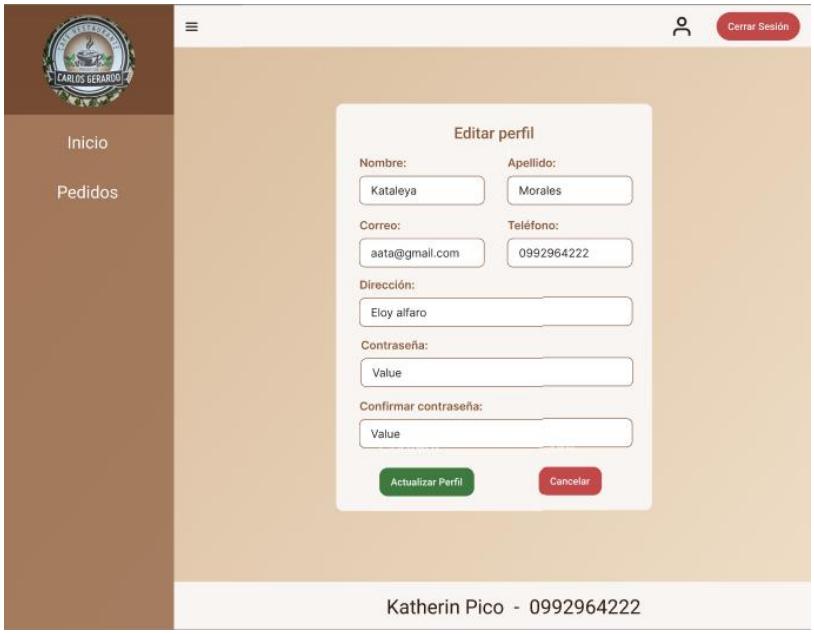
Se observa el formulario para editar la información del perfil del administrador.



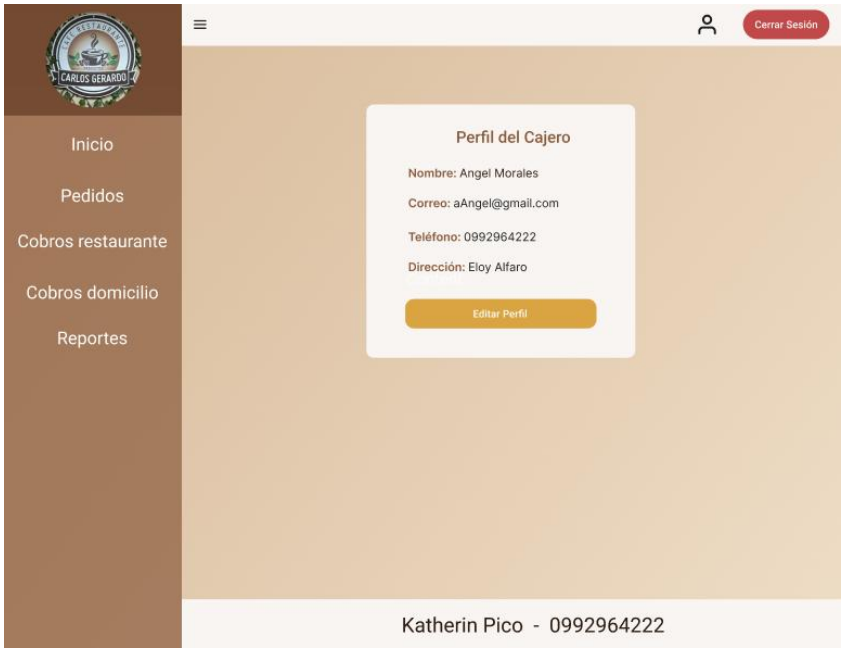
Se observa el perfil del mesero con sus datos personales.



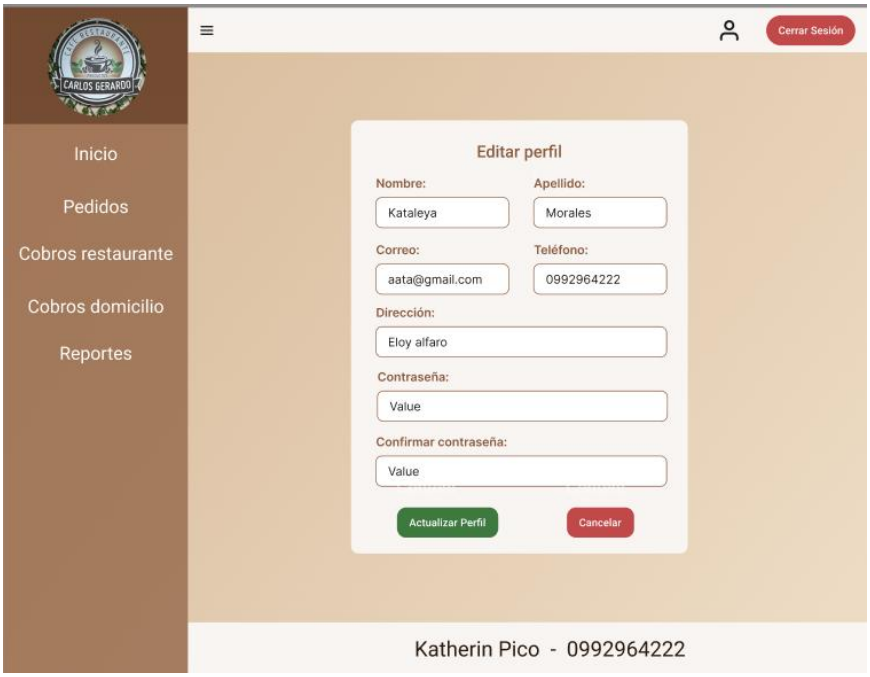
Se observa el formulario para editar la información del perfil del mesero.



Se observa el perfil del cajero con sus datos personales.



Se observa el formulario para editar la información del perfil del cajero.



CODIFICACIÓN

Se observa el código del views en donde se programa la lógica del backend visualizar y editar el perfil del cajero.

```python
# ----------------------------
# Cajero - Perfil
# ----------------------------
@login_required(login_url='login')
@rol_requerido('cajero')
def perfil_cajero(request):
    usuario = request.user   # cajero logueado
    return render(request, "cajero/perfil.html", {"usuario": usuario})

@login_required(login_url='login')
@rol_requerido('cajero')
def editar_perfil_cajero(request):
    usuario = request.user  # cajero logueado

    if request.method == "POST":
        nombre = request.POST.get("nombre", "").strip()
        apellido = request.POST.get("apellido", "").strip()
        correo = request.POST.get("correo", "").strip()
        telefono = request.POST.get("telefono", "").strip()
        direccion = request.POST.get("direccion", "").strip()
        password = request.POST.get("password", "")
        password2 = request.POST.get("password2", "")

        # ----------------------------
        # VALIDACIONES BACKEND
        # ----------------------------

        # Nombre y apellido: solo letras
        if not re.match(r'^[A-Za-zÁÉÍÓÚáéíóúÑñ\s]+$', nombre):
            messages.error(request, "El nombre solo debe contener letras.")
            return render(request, "cajero/editar_perfil.html", {"usuario": usuario})

        if not re.match(r'^[A-Za-zÁÉÍÓÚáéíóúÑñ\s]+$', apellido):
            messages.error(request, "El apellido solo debe contener letras.")
            return render(request, "cajero/editar_perfil.html", {"usuario": usuario})

        # Correo obligatorio + válido
        if not correo:
            messages.error(request, "El correo es obligatorio.")
            return render(request, "cajero/editar_perfil.html", {"usuario": usuario})

        if not re.match(r"^[^@]+@[^@]+\.[^@]+$", correo):
            messages.error(request, "El formato del correo no es válido.")
            return render(request, "cajero/editar_perfil.html", {"usuario": usuario})

```

Se evidencia el código del template en donde se va a mostrar la información personal del cajero.

```
<div class="perfil-container">
    <div class="perfil-card">

        <div class="perfil-header">
            <i class="fas fa-user-circle fa-4x" style="color: var(--cafe-oscuro);"></i>
            <h3>Perfil del Cajero</h3>
        </div>

        <div class="perfil-dato">
            <span>Nombre:</span> {{ usuario.nombre }} {{ usuario.apellido }}
        </div>

        <div class="perfil-dato">
            <span>Correo:</span> {{ usuario.correo }}
        </div>

        <div class="perfil-dato">
            <span>Teléfono:</span> {{ usuario.telefono|default:"No registrado" }}
        </div>

        <div class="perfil-dato">
            <span>Dirección:</span> {{ usuario.direccion|default:"No registrada" }}
        </div>

        <div class="perfil-dato">
            <span>Rol:</span> {{ usuario.get_rol_display }}
        </div>

        <a href="{% url 'editar_perfil_cajero' %}" class="btn btn-editar">
            <i class="fas fa-edit"></i> Editar Perfil
        </a>

    </div>
</div>
```

Se observa el código del views en donde se programa la lógica del backend visualizar y editar el perfil del mesero.

```python
#-----------------------------
#PERFIL - MESERO
#-----------------------------
@login_required
@rol_requerido('mesero')
def perfil_mesero(request):
    usuario = request.user  # si tu auth usa tu modelo Usuario como user
    return render(request, "mesero/perfil_mesero.html", {"usuario": usuario})


@login_required
@rol_requerido('mesero')
def editar_perfil_mesero(request):
    usuario = request.user

    if request.method == "POST":
        nombre = request.POST.get("nombre", "").strip()
        apellido = request.POST.get("apellido", "").strip()
        correo = request.POST.get("correo", "").strip()
        telefono = request.POST.get("telefono", "").strip()
        direccion = request.POST.get("direccion", "").strip()

        password = request.POST.get("password", "").strip()
        password2 = request.POST.get("password2", "").strip()

        # Actualizar datos
        usuario.nombre = nombre
        usuario.apellido = apellido
        usuario.correo = correo
        usuario.telefono = telefono
        usuario.direccion = direccion

        # Cambiar contraseña (solo si llenó)
        if password or password2:
            if password != password2:
                messages.error(request, "Las contraseñas no coinciden.")
                return redirect("editar_perfil_mesero")

            if len(password) < 6:
                messages.error(request, "La contraseña debe tener mínimo 6 caracteres.")
                return redirect("editar_perfil_mesero")

            usuario.password = make_password(password)

        usuario.save()
```

Se evidencia el código del template en donde se va a mostrar la información personal del mesero.

```
<div class="perfil-container">
    <div class="perfil-card">

        <div class="perfil-header">
            <i class="fas fa-user-circle fa-4x" style="color: var(--cafe-oscuro);"></i>
            <h3>Perfil del Mesero</h3>
        </div>

        <div class="perfil-dato">
            <span>Nombre:</span> {{ usuario.nombre }} {{ usuario.apellido }}
        </div>

        <div class="perfil-dato">
            <span>Correo:</span> {{ usuario.correo }}
        </div>

        <div class="perfil-dato">
            <span>Teléfono:</span> {{ usuario.telefono|default:"No registrado" }}
        </div>

        <div class="perfil-dato">
            <span>Dirección:</span> {{ usuario.direccion|default:"No registrada" }}
        </div>

        <div class="perfil-dato">
            <span>Rol:</span> {{ usuario.get_rol_display }}
        </div>

        <a href="{% url 'editar_perfil_mesero' %}" class="btn btn-editar">
            <i class="fas fa-edit"></i> Editar Perfil
        </a>

    </div>
```

Se observa el código del views en donde se programa la lógica del backend visualizar y editar el perfil del administrador.

```python
# ----------------------------
# Admin-Perfil
# ----------------------------
@login_required(login_url='login')
@rol_requerido('admin')
def perfil_admin(request):
    usuario = request.user   # usuario logueado
    return render(request, "administrador/perfil.html", {"usuario": usuario})

@login_required(login_url='login')
@rol_requerido('admin')
def editar_perfil_admin(request):
    usuario = request.user  # administrador logueado

    if request.method == "POST":
        nombre = request.POST.get("nombre", "").strip()
        apellido = request.POST.get("apellido", "").strip()
        correo = request.POST.get("correo", "").strip()
        telefono = request.POST.get("telefono", "").strip()
        direccion = request.POST.get("direccion", "").strip()
        password = request.POST.get("password", "")
        password2 = request.POST.get("password2", "")

        # ----------------------------
        # VALIDACIONES BACKEND
        # ----------------------------

        # Nombre y apellido: solo letras
        if not re.match(r'^[A-Za-zÁÉÍÓÚáéíóúÑñ\s]+$', nombre):
            messages.error(request, "El nombre solo debe contener letras.")
            return render(request, "administrador/editar_perfil_admin.html", {"usuario": usuario})

        if not re.match(r'^[A-Za-zÁÉÍÓÚáéíóúÑñ\s]+$', apellido):
            messages.error(request, "El apellido solo debe contener letras.")
            return render(request, "administrador/editar_perfil_admin.html", {"usuario": usuario})

        # Correo obligatorio + válido
        if not correo:
            messages.error(request, "El correo es obligatorio.")
            return render(request, "administrador/editar_perfil_admin.html", {"usuario": usuario})

        if not re.match(r"^[^@]+@[^@]+\.[^@]+$", correo):
            messages.error(request, "El formato del correo no es válido.")
            return render(request, "administrador/editar_perfil_admin.html", {"usuario": usuario})
```

Se evidencia el código del template en donde se va a mostrar la información personal del administrador.

```
<div class="perfil-container">
    <div class="perfil-card">

        <div class="perfil-header">
            <i class="fas fa-user-circle fa-4x" style="color: var(--cafe-oscuro);"></i>
            <h3>Perfil del Administrador</h3>
        </div>

        <div class="perfil-dato">
            <span>Nombre:</span> {{ usuario.nombre }} {{ usuario.apellido }}
        </div>

        <div class="perfil-dato">
            <span>Correo:</span> {{ usuario.correo }}
        </div>

        <div class="perfil-dato">
            <span>Teléfono:</span> {{ usuario.telefono|default:"No registrado" }}
        </div>

        <div class="perfil-dato">
            <span>Dirección:</span> {{ usuario.direccion|default:"No registrada" }}
        </div>

        <div class="perfil-dato">
            <span>Rol:</span> {{ usuario.get_rol_display }}
        </div>

        <a href="{%  url 'editar_perfil_admin' %}" class="btn btn-editar">
            <i class="fas fa-edit"></i> Editar Perfil
        </a>

    </div>
</div>
```

Se observa el código del views en donde se programa la lógica del backend visualizar y editar el perfil del cocinero.

```python
# ----------------------------
# Cocinero - Perfil
# ----------------------------
@login_required(login_url='login')
@rol_requerido('cocinero')
def perfil_cocinero(request):
    usuario = request.user
    return render(request, "cocinero/perfil/perfil_cocinero.html", {
        "usuario": usuario
    })


@login_required(login_url='login')
@rol_requerido('cocinero')
def editar_perfil_cocinero(request):
    usuario = request.user

    if request.method == "POST":
        nombre = request.POST.get("nombre", "").strip()
        apellido = request.POST.get("apellido", "").strip()
        correo = request.POST.get("correo", "").strip()
        telefono = request.POST.get("telefono", "").strip()
        direccion = request.POST.get("direccion", "").strip()

        password = request.POST.get("password", "").strip()
        password2 = request.POST.get("password2", "").strip()

        # ========================
        # VALIDACIONES BACKEND
        # ========================

        # Nombre y apellido solo letras y espacios (con tildes)
        patron_letras = r"^[A-Za-zÁÉÍÓÚáéíóúÑñ\s]+$"
        if not re.match(patron_letras, nombre):
            messages.error(request, "El nombre solo debe contener letras.")
            return render(request, "cocinero/perfil/editar_perfil_cocinero.html", {"usuario": usuario})

        if not re.match(patron_letras, apellido):
            messages.error(request, "El apellido solo debe contener letras.")
            return render(request, "cocinero/perfil/editar_perfil_cocinero.html", {"usuario": usuario})

        # Correo válido
        try:
            validate_email(correo)
        except ValidationError:
```

Se evidencia el código del template en donde se va a mostrar la información personal del cocinero.

```
<div class="perfil-container">
    <div class="perfil-card">

        <div class="perfil-header">
            <i class="fas fa-user-circle fa-4x" style="color: var(--cafe-oscuro);"></i>
            <h3>Perfil del Cocinero</h3>
        </div>

        <div class="perfil-dato">
            <span>Nombre:</span> {{ usuario.nombre }} {{ usuario.apellido }}
        </div>

        <div class="perfil-dato">
            <span>Correo:</span> {{ usuario.correo }}
        </div>

        <div class="perfil-dato">
            <span>Teléfono:</span> {{ usuario.telefono|default:"No registrado" }}
        </div>

        <div class="perfil-dato">
            <span>Dirección:</span> {{ usuario.direccion|default:"No registrada" }}
        </div>

        <div class="perfil-dato">
            <span>Rol:</span> {{ usuario.get_rol_display }}
        </div>

        <a href="{% url 'editar_perfil_cocinero' %}" class="btn btn-editar">
            <i class="fas fa-edit"></i> Editar Perfil
        </a>

    </div>
</div>
```

PRUEBAS

Caso de Prueba – HU12 Editar perfil administrador	
Código del Caso de Prueba	CP-40
Historia de Usuario	HU12 – Editar perfil administrador
Nombre del Caso de Prueba	Editar perfil administrador
Objetivo	Validar que el administrador pueda editar y actualizar su información personal en el sistema.
Usuario	Administrador

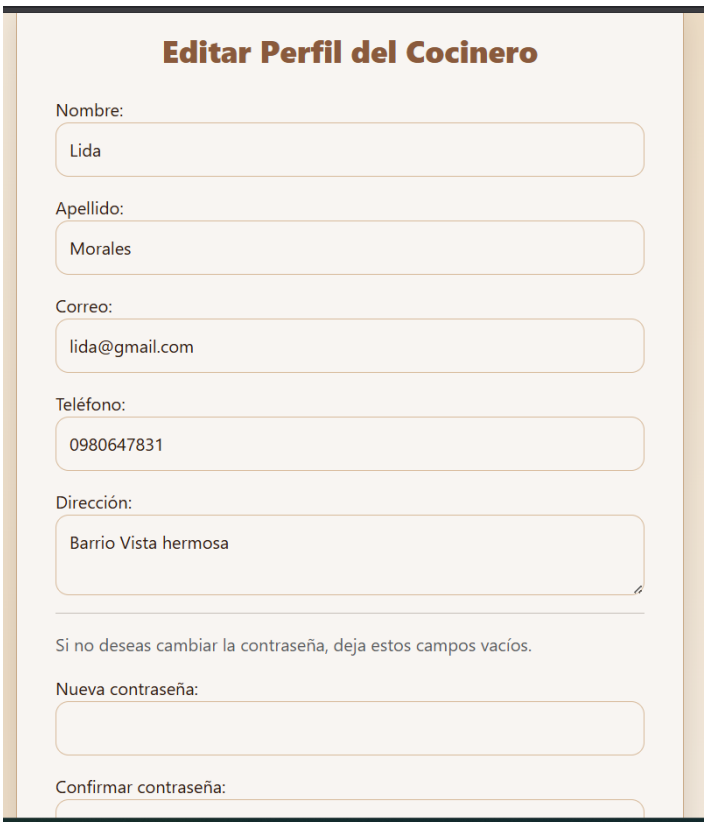
Precondiciones	El administrador debe haber iniciado sesión en el sistema.
Datos de Prueba	Nombre: Carlos Morales Correo: administrador@restaurante.com Teléfono: 0987654321
Pasos a Ejecutar	1. Ingresar al sistema como administrador. 2. Acceder al módulo de perfil. 3. Seleccionar la opción “Editar perfil”. 4. Modificar los datos personales. 5. Guardar los cambios.
Resultado Esperado	El sistema actualiza correctamente los datos del perfil del administrador.
Resultado Obtenido	El sistema guarda los cambios realizados en el perfil del administrador.
Evidencia	 

Caso de Prueba – HU13 Editar perfil mesero	
Código del Caso de Prueba	CP-41
Historia de Usuario	HU13 – Editar perfil mesero
Nombre del Caso de Prueba	Editar perfil mesero
Objetivo	Validar que el mesero pueda editar su información personal en el sistema.
Usuario	Mesero
Precondiciones	El mesero debe haber iniciado sesión en el sistema.
Datos de Prueba	Nombre: Luis Andrade Correo: mesero@restaurante.com Teléfono: 0991234567
Pasos a Ejecutar	1. Ingresar al sistema como mesero. 2. Acceder al módulo de perfil. 3. Seleccionar la opción “Editar perfil”. 4. Modificar los datos personales. 5. Guardar cambios.
Resultado Esperado	El sistema actualiza correctamente los datos del perfil del mesero.
Resultado Obtenido	El sistema guarda los cambios realizados en el perfil del mesero.

Evidencia	 

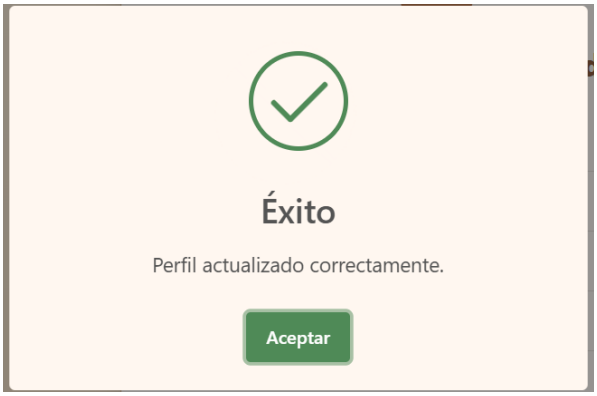
Caso de Prueba – HU24 Editar perfil cocinero	
Código del Caso de Prueba	CP-42
Historia de Usuario	HU24 – Editar perfil cocinero
Nombre del Caso de Prueba	Editar perfil cocinero
Objetivo	Validar que el cocinero pueda editar su información personal dentro del sistema.
Usuario	Cocinero

Precondiciones	El cocinero debe haber iniciado sesión en el sistema.
Datos de Prueba	Nombre: José Sánchez Correo: cocinero@restaurante.com Teléfono: 0981122334
Pasos a Ejecutar	1. Ingresar al sistema como cocinero. 2. Acceder al módulo de perfil. 3. Seleccionar la opción “Editar perfil”. 4. Modificar los datos personales. 5. Guardar cambios.
Resultado Esperado	El sistema actualiza correctamente la información del perfil del cocinero.
Resultado Obtenido	El sistema guarda los cambios realizados en el perfil del cocinero.

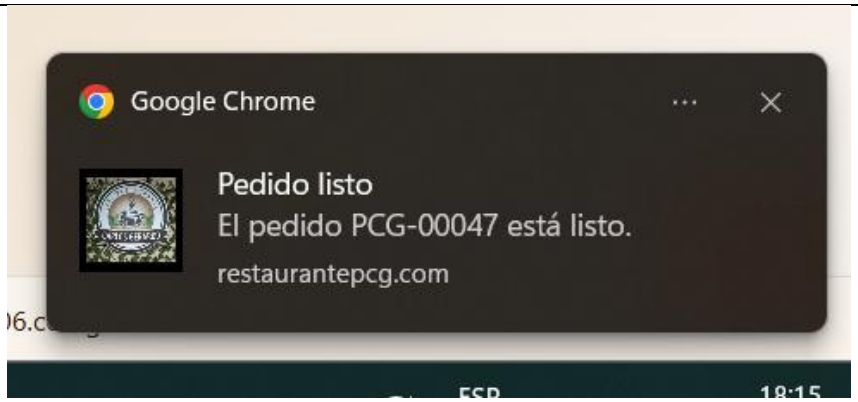
Evidencia	 

Caso de Prueba – HU25 Editar perfil cajero	
Código del Caso de Prueba	CP-43

Historia de Usuario	HU25 – Editar perfil cajero
Nombre del Caso de Prueba	Editar perfil cajero
Objetivo	Validar que el cajero pueda editar su información personal dentro del sistema.
Usuario	Cajero
Precondiciones	El cajero debe haber iniciado sesión en el sistema.
Datos de Prueba	Nombre: Ana Torres Correo: cajero@restaurante.com Teléfono: 0998877665
Pasos a Ejecutar	1. Ingresar al sistema como cajero. 2. Acceder al módulo de perfil. 3. Seleccionar la opción “Editar perfil”. 4. Modificar los datos personales. 5. Guardar cambios.
Resultado Esperado	El sistema actualiza correctamente los datos del perfil del cajero.
Resultado Obtenido	El sistema guarda los cambios realizados en el perfil del cajero.

Evidencia	 

Caso de Prueba – HU33 Recibir notificaciones	
Código del Caso de Prueba	CP-44
Historia de Usuario	HU33 – Recibir notificaciones
Nombre del Caso de Prueba	Recibir notificaciones del sistema
Objetivo	Validar que el personal del restaurante reciba notificaciones generadas por el sistema.
Usuario	Personal del restaurante

Precondiciones	El usuario debe haber iniciado sesión y existir eventos que generen notificaciones.
Datos de Prueba	Notificación: Pedido listo para entrega
Pasos a Ejecutar	1. Ingresar al sistema con un usuario del personal. 2. Generar una acción que produzca una notificación (por ejemplo, pedido listo). 3. Verificar la recepción de la notificación en el sistema.
Resultado Esperado	El sistema muestra una notificación al usuario en tiempo real.
Resultado Obtenido	El sistema envía y muestra la notificación correctamente al usuario.
Evidencia	

Caso de Prueba – HU34 Uso en dispositivos móviles	
Código del Caso de Prueba	CP-45
Historia de Usuario	HU34 – Uso en dispositivos móviles

Nombre del Caso de Prueba	Uso del sistema en dispositivos móviles
Objetivo	Validar que el sistema funcione correctamente en dispositivos móviles.
Usuario	Personal del restaurante
Precondiciones	El usuario debe tener acceso al sistema desde un dispositivo móvil con conexión a internet.
Datos de Prueba	Dispositivo: Teléfono móvil Navegador: Chrome móvil
Pasos a Ejecutar	1. Abrir el navegador desde un dispositivo móvil. 2. Ingresar a la URL del sistema. 3. Iniciar sesión con un usuario del sistema. 4. Navegar por los diferentes módulos del sistema.
Resultado Esperado	El sistema se visualiza correctamente y permite realizar las funciones principales desde el dispositivo móvil.
Resultado Obtenido	El sistema funciona correctamente en dispositivos móviles y permite acceder a sus funcionalidades.

Evidencia	

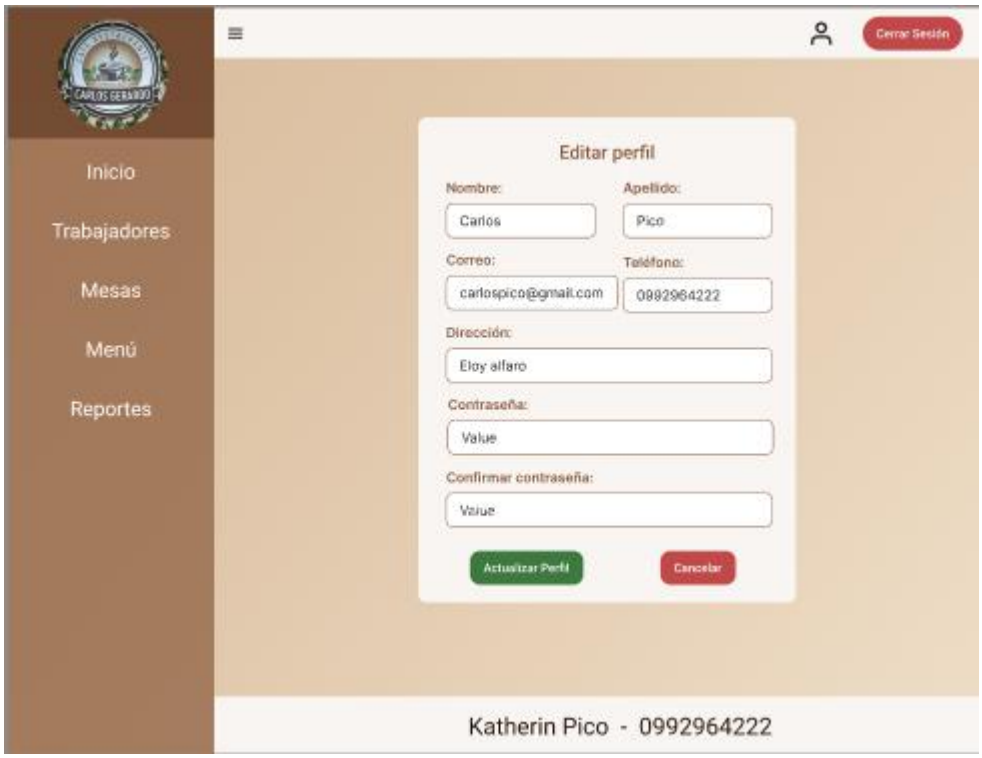
ANEXO K: ITERACIÓN 10 – MÓDULO CLIENTE (PEDIDOS EN LÍNEA)

PLANIFICACIÓN

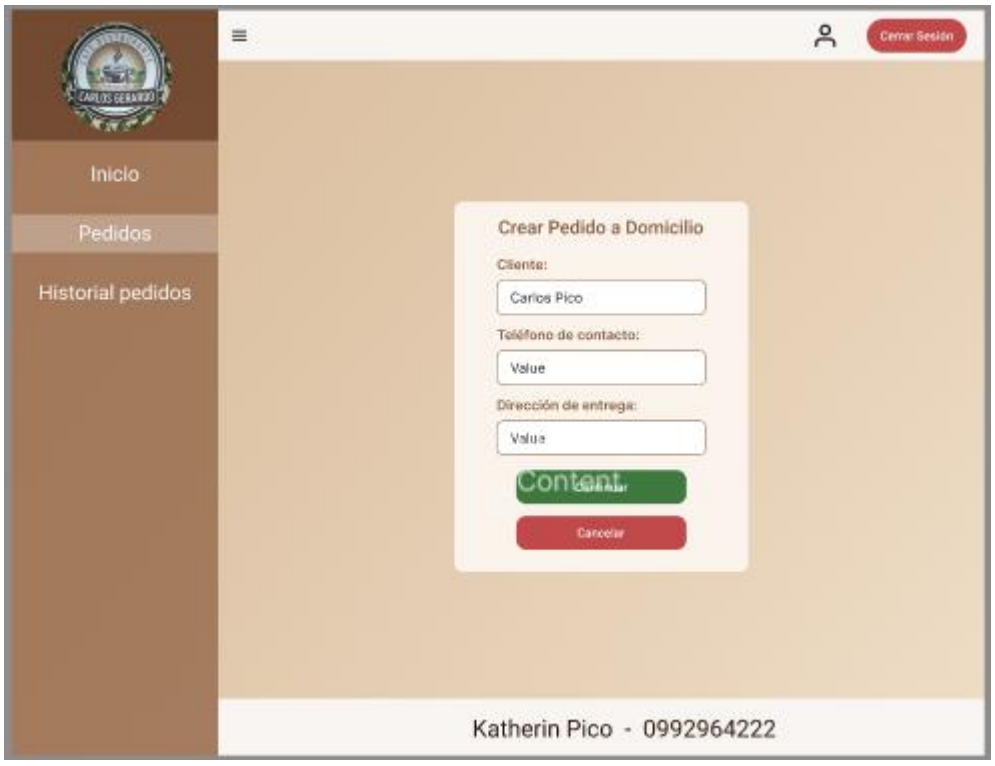
Número	Historia de Usuario
HU46	Registro de cliente
HU47	Editar perfil cliente
HU48	Crear pedido
HU49	Agregar productos
HU50	Subir comprobante
HU51	Enviar pedido al cajero

DISEÑO

Se observa el formulario para editar la información del perfil del cliente.

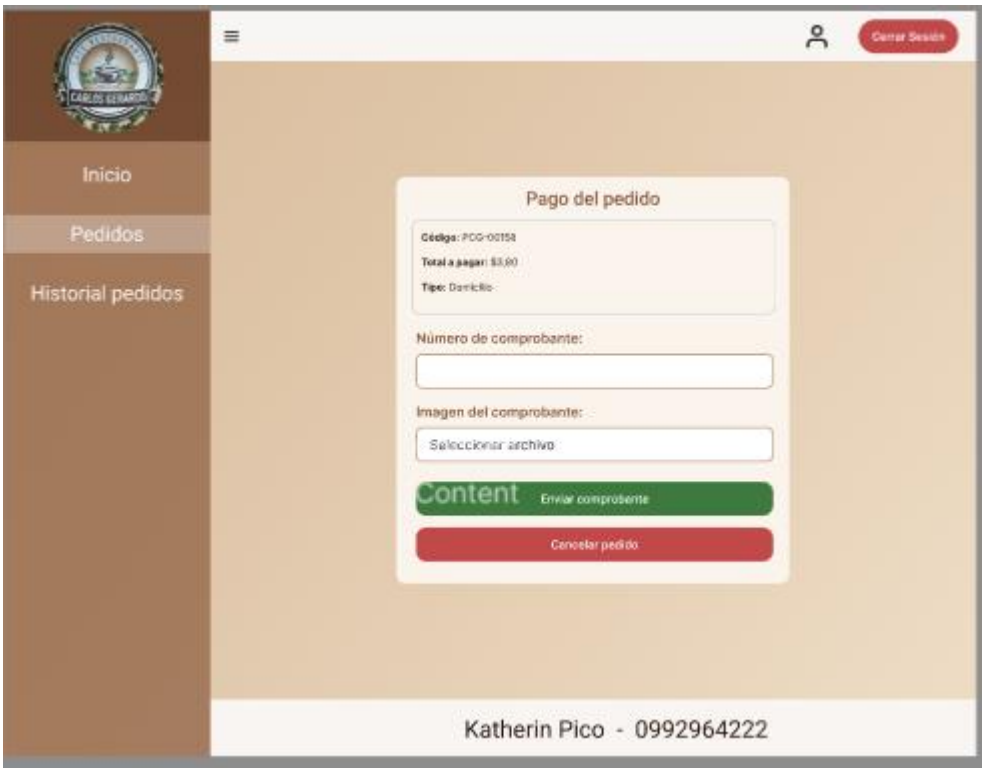


Se observa el formulario para crear un pedido a domicilio por parte del cliente.



Se observa el formulario para agregar productos al pedido creado por el cliente.

Se observa el formulario para agregar el número del comprobante y la imagen del comprobante de la transferencia para registrar el pago del pedido.



Se observa que el pedido ya se realizó y fue enviado al cajero para su revisión.

CODIFICACIÓN

Se evidencia el código del template en donde se va a mostrar el formulario para editar el perfil del cliente.

Aplicacion > gestionPedidos > templates > cliente > perfil > editar_perfil.html > style

```
</style>

<div class="form-container">
    <div class="card-form">

        <form id="frm_cliente" method="post" novalidate>
            {% csrf_token %}

            <h2 class="titulo-form">Editar Perfil del Cliente</h2>

            <div class="mb-3">
                <label>Nombre:</label>
                <input type="text" name="nombre" class="form-control"
                       value="{{ usuario.nombre }}" required>
            </div>

            <div class="mb-3">
                <label>Apellido:</label>
                <input type="text" name="apellido" class="form-control"
                       value="{{ usuario.apellido }}" required>
            </div>

            <div class="mb-3">
                <label>Correo:</label>
                <input type="email" name="correo" class="form-control"
                       value="{{ usuario.correo }}" required>
            </div>

            <div class="mb-3">
                <label>Teléfono:</label>
                <input type="text" name="telefono" class="form-control"
                       value="{{ usuario.telefono }}" required>
            </div>

            <div class="mb-3">
                <label>Dirección:</label>
                <textarea name="direccion" class="form-control"
                          rows="2" required>{{ usuario.direccion }}</textarea>
            </div>

            <hr>

            <p class="text-muted">
                Si no deseas cambiar la contraseña, deja estos campos vacíos.
            </p>
```

Se evidencia el código del template en donde se va a mostrar el formulario para crear un pedido a domicilio por el cliente.

Aplicacion > gestionPedidos > templates > cliente > pedidos > crear_pedido.html > ...

```
<div class="main-content-wrapper">

<div class="form-container">

    <div class="card-form">

        <h2 class="titulo-form">Crear Pedido a Domicilio</h2>

        {% if messages %}
            {% for message in messages %}
                <script>
                    document.addEventListener("DOMContentLoaded", function () {
                        Swal.fire({
                            icon: "{% if message.tags == 'error' %}error{% else %}success{% endif %}",
                            title: "{% if message.tags == 'error' %}¡Atención!{% else %}Éxito{% endif %}",
                            text: "{{ message|escapejs }}",
                            confirmButtonText: "Aceptar",
                            confirmButtonColor: "{% if message.tags == 'error' %}#C14949{% else %}#4F8A57{% endif %}",
                            background: "#FFF7F0"
                        });
                    });
                </script>
            {% endfor %}
        {% endif %}

        <form method="POST">
            {% csrf_token %}

            <div class="mb-3">
                <label>Cliente:</label>
                <input type="text"
                       class="form-control"
                       value="{{ cliente.nombre }} {{ cliente.apellido }}"
                       readonly>
            </div>

            <div class="mb-3">
                <label>Teléfono de contacto:</label>
                <input type="text"
                       name="telefono"
                       id="telefono"
                       class="form-control"
                       placeholder="Ej: 0991234567">
                <div class="error-text" id="error_telefono"></div>
            </div>
```

Se evidencia el código del template en donde se va a mostrar el formulario para agregar los productos al pedido creado por el cliente.

Aplicacion > gestionPedidos > templates > cliente > pedidos > agregar_detalles.html > ...

```
<div class="main-content-wrapper">
    <div class="container mt-5 content-card">

        <h2 class="text-center titulo-pedido mb-4">
            Agregar Productos al Pedido {{ pedido.codigo_pedido }}
        </h2>

        <!-- GRID DE PRODUCTOS -->
        <div class="menu-grid">
            {% for producto in productos %}
            <div>
                <div class="card card-producto p-3 h-100 {% if producto.agotado_hoy %}producto-agotado{% endif %}
                    id="producto-{{ producto.id }}">


                    <h5 class="fw-bold text-center mb-2">
                        {{ producto.nombre }}
                        {% if producto.agotado_hoy %}
                        <span class="badge bg-secondary ms-2">Agotado hoy</span>
                        {% endif %}
                    </h5>

                    <p class="text-muted text-center small mb-2">
                        {{ producto.descripcion }}
                    </p>

                    <p class="fw-bold text-success text-center mb-2">
                        ${{ producto.precio }}
                    </p>

                    <label>Cantidad</label>
                    <input type="number" min="1" value="1"
                        id="cantidad_{{ producto.id }}"
                        class="form-control mb-2"
                        {% if producto.agotado_hoy %}disabled{% endif %}>

                    <label>Observación</label>
                    <input type="text"
                        id="obs_{{ producto.id }}"
                        class="form-control mb-2"
                        {% if producto.agotado_hoy %}disabled{% endif %}>

                    <button class="btn btn-agregar w-100"
                            data-producto-id="{{ producto.id }}"
                            {% if producto.agotado_hoy %}disabled{% endif %}
```

Se evidencia el código del template en donde se va a mostrar el formulario para registrar el pago del pedido realizado por el cliente.

Aplicacion > gestionPedidos > templates > cliente > pedidos > pago.html > ...

```
<div class="form-container">

    <div class="card-form">

        <h2 class="titulo-form">Pago del Pedido</h2>

        <!-- RESUMEN -->
        <div class="resumen-box">
            <p><strong>Código:</strong> {{ pedido.codigo_pedido }}</p>
            <p><strong>Total a pagar:</strong> ${{ pedido.total }}</p>
            <p><strong>Tipo:</strong> Domicilio</p>
        </div>

        <!-- FORMULARIO -->
        <form method="POST" enctype="multipart/form-data">
            {% csrf_token %}

            <div class="mb-3">
                <label>Número de comprobante</label>
                <input type="text"
                       name="numero_comprobante"
                       class="form-control"
                       required>
            </div>

            <div class="mb-3">
                <label>Imagen del comprobante</label>
                <input type="file"
                    name="imagen"
                    class="form-control"
                    accept=".jpg,.jpeg,.png,.webp"
                    required>
            </div>

            <button class="btn btn-guardar w-100 mt-4">
                Enviar comprobante
            </button>

            <button type="button"
                    class="btn btn-cancelar"
                    onclick="cancelarPedido('{{ pedido.id }}')">
                Cancelar pedido
            </button>
        </form>

```

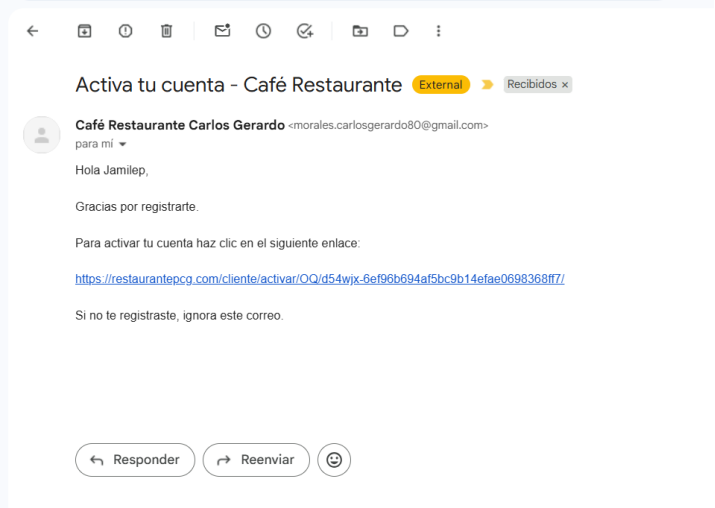
Se evidencia el código del template en donde se va a mostrar el detalle del pedido ya enviado al cajero para su revisión.

Aplicacion > gestionPedidos > templates > cliente > pedidos > ver_pedido.html > ...

```
<div class="main-content-wrapper">
    <div class="container mt-5">

        <h2 class="titulo-pedido">
            Detalle del Pedido a Domicilio
        </h2>

        <div class="mb-3">
            <a href="{% url 'cliente_listar_pedidos' %}" class="btn btn-regresar">
                <i class="fas fa-arrow-left me-2"></i> Regresar
            </a>
        </div>

        <!-- RESUMEN PEDIDO + PAGO -->
        <div class="resumen-pedido mb-4">

            <!-- HEADER -->
            <div class="resumen-header">
                <div class="resumen-codigo">
                    {{ pedido.codigo_pedido }}
                </div>

                <div>
                    {% if pedido.estado == 'en_creacion' %}
                        <span class="badge badge-creacion">En creación</span>
                    {% elif pedido.estado == 'aceptado' %}
                        <span class="badge bg-info">Aceptado</span>
                    {% elif pedido.estado == 'en preparacion' %}
                        <span class="badge bg-warning">En preparación</span>
                    {% elif pedido.estado == 'listo' %}
                        <span class="badge bg-success">Listo</span>
                    {% elif pedido.estado == 'finalizado' %}
                        <span class="badge bg-primary">Finalizado</span>
                    {% endif %}
                </div>

                <div class="resumen-total">
                    ${{ pedido.total|floatformat:2 }}
                </div>
            </div>

            <!-- DATOS DEL PEDIDO -->
            <div class="resumen-grid">
                <div class="resumen-item">
                    <small>Cliente</small>
```

PRUEBAS

Caso de Prueba – HU46 Registrarse en el sistema	
Código del Caso de Prueba	CP-46
Historia de Usuario	HU46 – Registrarse en el sistema
Nombre del Caso de Prueba	Registro de cliente en el sistema
Objetivo	Validar que un cliente pueda registrarse en el sistema ingresando sus datos personales.
Usuario	Cliente
Precondiciones	El cliente debe tener acceso a la plataforma del sistema.
Datos de Prueba	Nombre: María López Correo: maria@gmail.com Teléfono: 0999988776 Contraseña: Maria123
Pasos a Ejecutar	1. Acceder a la página principal del sistema. 2. Seleccionar la opción “Registrarse”. 3. Ingresar los datos personales solicitados. 4. Presionar el botón “Crear cuenta”.
Resultado Esperado	El sistema registra correctamente al cliente y crea una cuenta de usuario.
Resultado Obtenido	El sistema guarda los datos del cliente y permite el acceso al sistema.

Evidencia	  

Caso de Prueba – HU47 Editar perfil	
Código del Caso de Prueba	CP-47
Historia de Usuario	HU47 – Editar perfil
Nombre del Caso de Prueba	Editar perfil del cliente
Objetivo	Validar que el cliente pueda modificar su información personal dentro del sistema.
Usuario	Cliente
Precondiciones	El cliente debe haber iniciado sesión en el sistema.
Datos de Prueba	Nombre actualizado: María López Torres Teléfono actualizado: 0994455667
Pasos a Ejecutar	1. Ingresar al sistema con el usuario cliente. 2. Acceder al menú de perfil. 3. Seleccionar la opción “Editar perfil”.

	4. Modificar los datos personales. 5. Guardar los cambios.
Resultado Esperado	El sistema actualiza correctamente la información del cliente.
Resultado Obtenido	El sistema guarda los cambios realizados en el perfil del cliente.
Evidencia	 

Caso de Prueba – HU48 Crear pedido	
Código del Caso de Prueba	CP-48
Historia de Usuario	HU48 – Crear pedido

Nombre del Caso de Prueba	Crear pedido desde la plataforma
Objetivo	Validar que el cliente pueda crear un pedido en la plataforma del restaurante.
Usuario	Cliente
Precondiciones	El cliente debe haber iniciado sesión en el sistema.
Datos de Prueba	Cliente: María López Tipo de pedido: Domicilio
Pasos a Ejecutar	1. Ingresar al sistema como cliente. 2. Acceder al módulo de pedidos. 3. Seleccionar la opción “Nuevo pedido”. 4. Confirmar la creación del pedido.
Resultado Esperado	El sistema crea correctamente un nuevo pedido para el cliente.
Resultado Obtenido	El sistema registra el pedido en estado pendiente.
Evidencia	

Caso de Prueba – HU49 Agregar productos al pedido	
Código del Caso de Prueba	CP-49
Historia de Usuario	HU49 – Agregar productos al pedido
Nombre del Caso de Prueba	Agregar productos al pedido
Objetivo	Validar que el cliente pueda agregar productos al pedido indicando cantidad.
Usuario	Cliente
Precondiciones	El cliente debe haber iniciado sesión y tener un pedido creado.
Datos de Prueba	Producto: Bolón completo Cantidad: 2 Precio unitario: $5.00
Pasos a Ejecutar	1. Ingresar al sistema como cliente. 2. Acceder al pedido creado. 3. Seleccionar un producto del menú. 4. Indicar la cantidad deseada. 5. Agregar el producto al pedido.
Resultado Esperado	El sistema agrega correctamente el producto al pedido del cliente.
Resultado Obtenido	El sistema registra el producto seleccionado con la cantidad indicada.

Evidencia	 

Caso de Prueba – HU50 Subir comprobante de pago	
Código del Caso de Prueba	CP-50
Historia de Usuario	HU50 – Subir comprobante de pago
Nombre del Caso de Prueba	Subir comprobante de transferencia bancaria
Objetivo	Validar que el cliente pueda subir un comprobante de pago para validar su pedido.
Usuario	Cliente
Precondiciones	El cliente debe haber creado un pedido y tener acceso al comprobante de transferencia.

Datos de Prueba	Archivo: comprobante_pago.jpg Monto: $5.00
Pasos a Ejecutar	1. Ingresar al sistema como cliente. 2. Acceder al pedido realizado. 3. Seleccionar la opción “Subir comprobante”. 4. Adjuntar el archivo del comprobante. 5. Guardar la información.
Resultado Esperado	El sistema guarda correctamente el comprobante de pago del cliente.
Resultado Obtenido	El sistema registra el comprobante adjunto al pedido.
Evidencia	 

Caso de Prueba – HU51 Enviar pedido para validación	
Código del Caso de Prueba	CP-51
Historia de Usuario	HU51 – Enviar pedido para validación
Nombre del Caso de Prueba	Enviar pedido para validación del cajero
Objetivo	Validar que el cliente pueda enviar su pedido para que el cajero valide el pago y procese la orden.
Usuario	Cliente
Precondiciones	El cliente debe haber creado un pedido y subido el comprobante de pago.
Datos de Prueba	Pedido N°: 120 Estado inicial: Pendiente de validación
Pasos a Ejecutar	1. Ingresar al sistema como cliente. 2. Acceder al pedido realizado. 3. Verificar que el comprobante esté cargado. 4. Seleccionar la opción “Enviar pedido para validación”.
Resultado Esperado	El sistema envía el pedido al cajero para su validación.
Resultado Obtenido	El sistema cambia el estado del pedido a “Pendiente de validación por cajero”.
Evidencia	

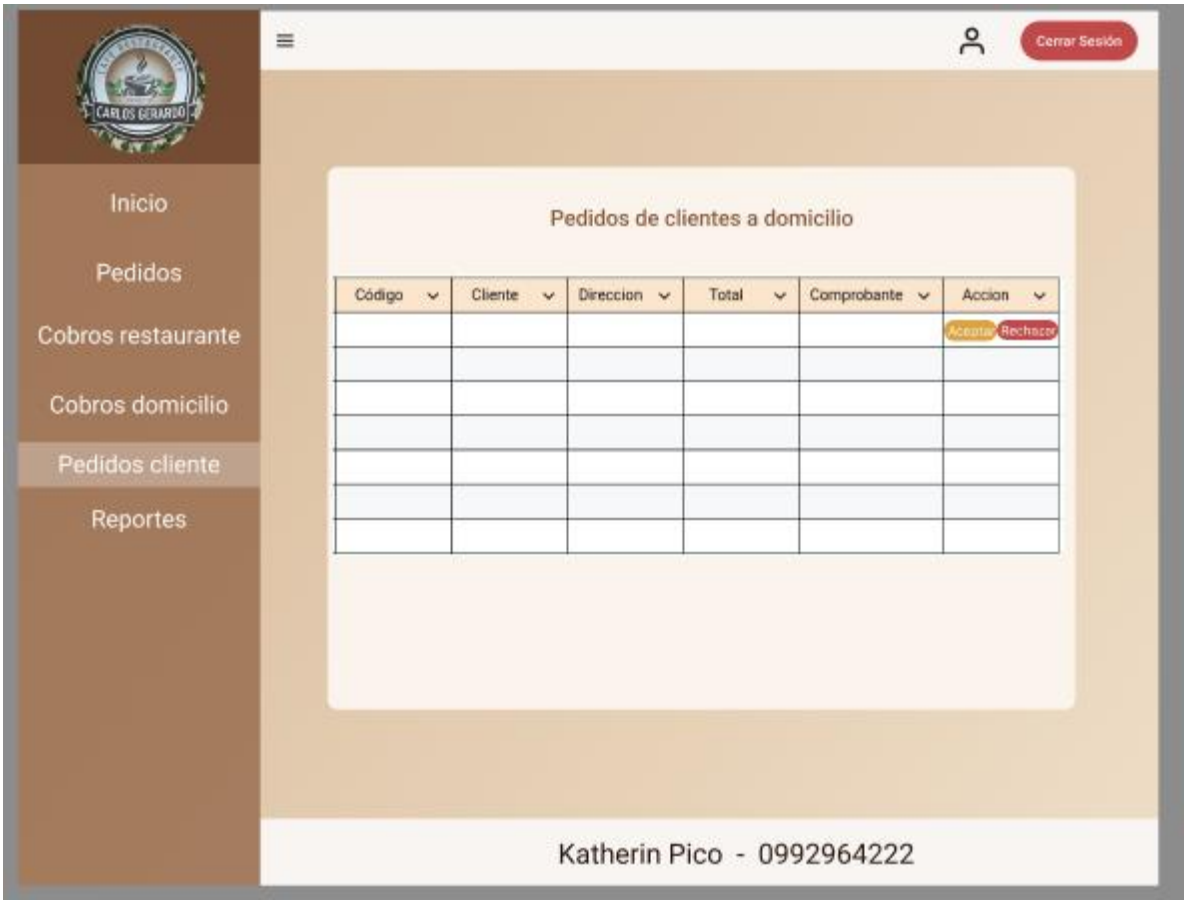
ANEXO L: ITERACIÓN 11 – VALIDACIÓN DE PAGOS EN LÍNEA

PLANIFICACIÓN

Número	Historia de Usuario
HU52	Validar comprobante
HU53	Aprobar pedido
HU54	Rechazar pedido

DISEÑO

Se observa la tabla en donde llegaran los pedidos realizados por el cliente para su respetiva verificación y así aceptar o rechazar dicho pedido.



CODIFICACIÓN

Se evidencia el código del template en donde se va a mostrar el pedido enviado por el cliente para su revisión y así ser aceptado o rechazado.

Aplicacion > gestionPedidos > templates > cajero > pedidos > pedidos_aceptados_rechazados.html > ...

```
<div class="main-content-wrapper">
    <div class="content-card">

        <h2 class="text-center mb-4">
            Historial de pedidos del cliente
        </h2>

        <div class="d-flex justify-content-between align-items-center mb-3">
            <a href="{% url 'cajero_reportes' %}" class="btn btn-regresar">
                Regresar
            </a>
        </div>

        <br><br>

        <!-- ================ FILTROS ================ -->
        <form method="get" class="row g-3 form-filtros mb-4 align-items-end">

            <div class="col-md-3">
                <label class="fw-semibold">Código de pedido</label>
                <div class="input-group">
                    <span class="input-group-text bg-white">
                        <i class="fas fa-search text-muted"></i>
                    </span>
                    <input type="text"
                        name="codigo"
                        class="form-control"
                        placeholder="PCG-00082"
                        value="{{ codigo|default:'' }}">
                </div>
            </div>

            <div class="col-md-3">
                <label class="fw-semibold">Estado del pedido</label>
                <select name="estado" class="form-select">
                    <option value="">Todos</option>
                    <option value="listo" {% if estado == 'listo' %}selected{% endif %}>
                        Listo
                    </option>
                    <option value="rechazado" {% if estado == 'rechazado' %}selected{% endif %}>
                        Rechazado
                    </option>
                </select>
            </div>

```

Se evidencia el código del views en donde se programó la lógica de aceptar un pedido realizado por el cliente.

```python
@login_required(login_url='login')
@rol_requerido('cajero')
@require_POST
def cajero_aceptar_pedido_cliente(request, pedido_id):

    pedido = get_object_or_404(
        Pedido,
        id=pedido_id,
        tipo_pedido='domicilio'
    )

    if pedido.estado != 'pendiente_caja':
        return JsonResponse({
            "success": False,
            "message": "Este pedido ya fue procesado."
        })

    # ================================
    # 1) ACEPTAR PEDIDO
    # ================================
    pedido.cajero = request.user
    pedido.estado = 'aceptado'
    pedido.enviado_cocina = True
    pedido.save(update_fields=['cajero', 'estado', 'enviado_cocina'])

    # ================================
    # DEFINIR DATOS DEL COMPROBANTE
    # ================================
    numero = f"DC-{pedido.id}"  # o el formato que tú quieras

    # Para pedidos de cliente (domicilio)
    nombre_comp = pedido.cliente_nombre
    direccion_comp = pedido.direccion_entrega or ""

    # ================================
    # 2) CREAR PAGO + COMPROBANTE
    # ================================
    with transaction.atomic():
        pago = Pago.objects.create(
            pedido=pedido,
            total=Decimal(str(pedido.total)),
            monto_recibido=Decimal(str(pedido.total)),
            metodo_pago="transferencia",
            estado_pago="confirmado"
        )
```

Se evidencia el código del views en donde se programó la lógica de rechazar un pedido realizado por el cliente.

```python
@login_required(login_url='login')
@rol_requerido('cajero')
@require_POST
def cajero_rechazar_pedido_cliente(request, pedido_id):

    pedido = get_object_or_404(
        Pedido,
        id=pedido_id,
        tipo_pedido='domicilio'
    )

    if pedido.estado != 'pendiente_caja':
        return JsonResponse({
            "success": False,
            "message": "Este pedido ya fue procesado."
        })

    comprobante = pedido.comprobantes_cliente.filter(estado='pendiente').first()
    if comprobante:
        comprobante.estado = 'rechazado'
        comprobante.save(update_fields=['estado'])

    pedido.estado = 'rechazado'
    pedido.enviado_cocina = False
    pedido.cajero = request.user
    pedido.save(update_fields=['estado', 'enviado_cocina', 'cajero'])

    channel_layer = get_channel_layer()

    # WS actualizar estado cliente
    async_to_sync(channel_layer.group_send)(
        f"estado_pedidos_cliente_{pedido.cliente.id}",
        {
            "type": "actualizar_estado",
            "pedido": pedido.id,
            "estado": "rechazado"
        }
    )

    # Notificación DB
    notif = Notificacion.objects.create(
        pedido=pedido,
        usuario_destino=pedido.cliente,
        tipo="pedido_rechazado",
        mensaje=f"Tu pedido #{pedido.codigo_pedido} fue rechazado."
```

PRUEBAS

Caso de Prueba – HU52 Validar comprobante de pago	
Código del Caso de Prueba	CP-52
Historia de Usuario	HU52 – Validar comprobante de pago
Nombre del Caso de Prueba	Validar comprobante de pago enviado por el cliente
Objetivo	Validar que el cajero pueda revisar el comprobante de transferencia enviado por el cliente para verificar que el pago sea correcto.
Usuario	Cajero
Precondiciones	El cajero debe haber iniciado sesión y existir un pedido con comprobante de pago enviado por el cliente.
Datos de Prueba	Pedido N°: 125 Cliente: María López Archivo comprobante: comprobante_transferencia.jpg Monto transferido: $12.00
Pasos a Ejecutar	1. Ingresar al sistema como cajero. 2. Acceder al módulo de pedidos en línea. 3. Seleccionar un pedido pendiente de validación. 4. Visualizar el comprobante de pago adjunto. 5. Verificar que el monto coincida con el total del pedido.
Resultado Esperado	El sistema permite visualizar correctamente el comprobante de pago enviado por el cliente.
Resultado Obtenido	El cajero revisa el comprobante de pago y confirma si el monto es correcto.

Evidencia	 

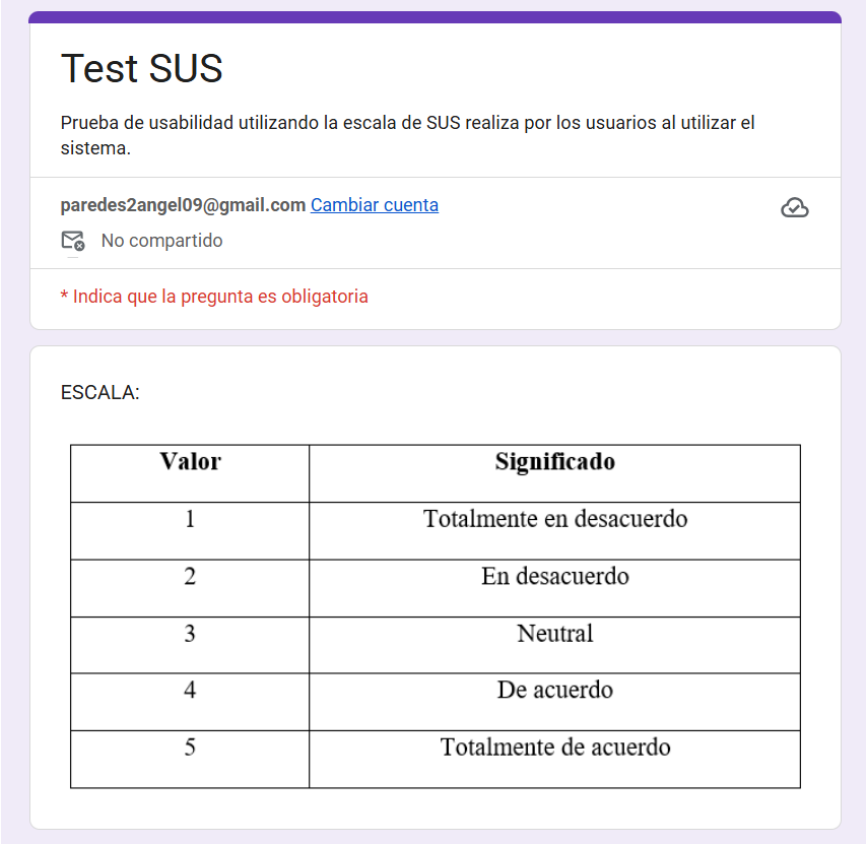
Caso de Prueba – HU53 Aprobar pedido en línea	
Código del Caso de Prueba	CP-53
Historia de Usuario	HU53 – Aprobar pedido en línea
Nombre del Caso de Prueba	Aprobar pedido en línea con pago válido

Objetivo	Validar que el cajero pueda aprobar un pedido en línea cuando el pago es válido.
Usuario	Cajero
Precondiciones	El cajero debe haber iniciado sesión y existir un pedido con comprobante de pago válido.
Datos de Prueba	Pedido N°: 126 Estado inicial: Pendiente de validación Monto pagado: $5.00
Pasos a Ejecutar	1. Ingresar al sistema como cajero. 2. Acceder al módulo de pedidos en línea. 3. Seleccionar un pedido pendiente de validación. 4. Revisar el comprobante de pago. 5. Presionar la opción “Aprobar pedido”.
Resultado Esperado	El sistema aprueba el pedido y lo envía automáticamente al área de cocina para su preparación.
Resultado Obtenido	El sistema cambia el estado del pedido a “Aprobado” y lo envía al módulo de cocina.
Evidencia	

Caso de Prueba – HU54 Rechazar pedido en línea	
Código del Caso de Prueba	CP-54
Historia de Usuario	HU54 – Rechazar pedido en línea
Nombre del Caso de Prueba	Rechazar pedido en línea por pago incorrecto
Objetivo	Validar que el cajero pueda rechazar un pedido cuando el comprobante de pago no es válido.
Usuario	Cajero
Precondiciones	El cajero debe haber iniciado sesión y existir un pedido con comprobante de pago incorrecto o inválido.
Datos de Prueba	Pedido N°: 127 Cliente: Juan Pérez Monto del pedido: $15.00 Monto transferido: $10.00
Pasos a Ejecutar	1. Ingresar al sistema como cajero. 2. Acceder al módulo de pedidos en línea. 3. Seleccionar un pedido pendiente de validación. 4. Revisar el comprobante de pago. 5. Presionar la opción “Rechazar pedido”.
Resultado Esperado	El sistema rechaza el pedido y notifica al cliente que el pago no fue válido.
Resultado Obtenido	El sistema cambia el estado del pedido a “Rechazado” y notifica al cliente.

Evidencia	 

ANEXO M: TEST SUS



Test SUS

Prueba de usabilidad utilizando la escala de SUS realiza por los usuarios al utilizar el sistema.

paredes2angel09@gmail.com Cambiar cuenta

No compartido

* Indica que la pregunta es obligatoria

ESCALA:

Valor	Significado
1	Totalmente en desacuerdo
2	En desacuerdo
3	Neutral
4	De acuerdo
5	Totalmente de acuerdo

1. Creo que me gustaría usar este sistema frecuentemente. *

1 2 3 4 5

2. Encontré el sistema innecesariamente complejo. *

1 2 3 4 5

3. Pensé que el sistema era fácil de usar. *

1 2 3 4 5

4. Creo que necesitaría ayuda técnica para usar este sistema. *

1 2 3 4 5

5. Las funciones del sistema están bien integradas. *

1 2 3 4 5

6. Encontré inconsistencias en el sistema. *

1 2 3 4 5

7. Creo que la mayoría de las personas aprenderían a usarlo rápidamente. *

1 2 3 4 5

8. El sistema me resultó complicado de usar. *

1 2 3 4 5

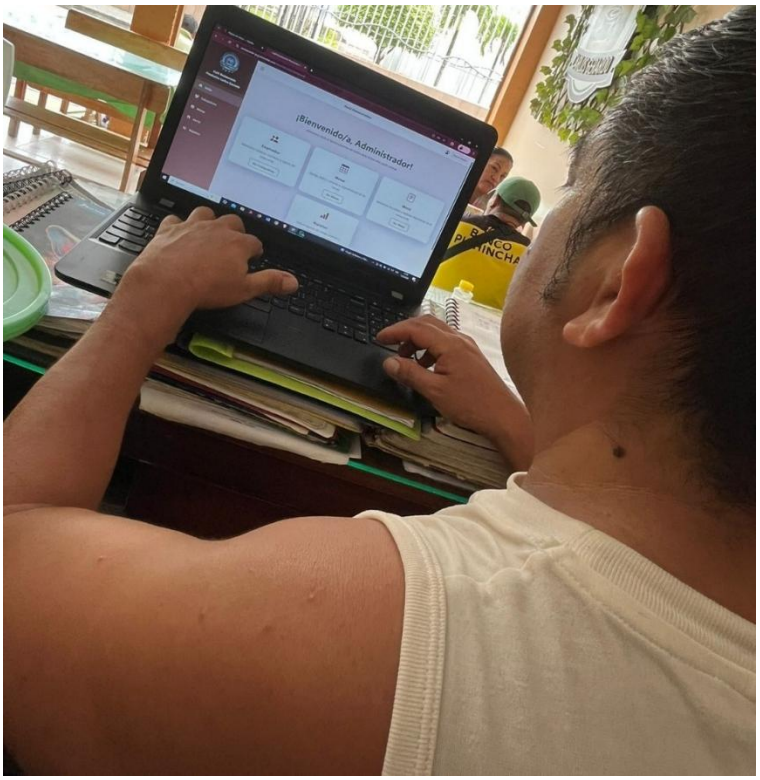
9. Me sentí seguro al usar el sistema. *

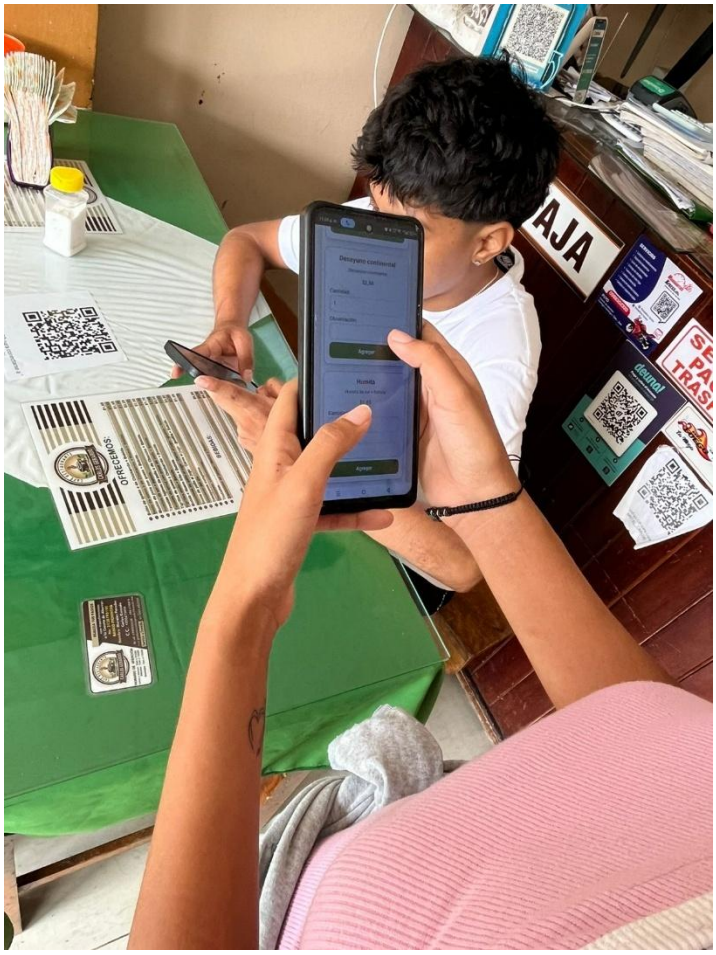
1 2 3 4 5

10. Necesité aprender muchas cosas antes de poder usarlo. *

1 2 3 4 5

ANEXO N: UTILIZACION DEL SISTEMA POR LOS USUARIOS FINALES




AJA