DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA PARA LA GESTIÓN DE UN FOOD TRUCK COMO PUNTO DE VENTA”

Informe final para optar el título de Ingeniero en Informática

Información General

Escuela	Informática y Telecomunicaciones
Carrera	Ingeniería en Informática
Sede	Plaza Oeste
Docente	Cristian Espinoza Silva

Integrantes

Rut	Nombre	Correo
16.808.385-8	Gustavo Pezzini Puen	gu.pezzini@duocuc.cl
11.867.330-1	René Veloso Salazar	re.veloso@duocuc.cl
21.078.145-5	Vicente Gálvez Roldán	vic.galvez@duocuc.cl

Histórico de Versiones

Versión	Fecha	Descripción/cambio
1	31/08/2025	Versión inicial
2	17/11/2025	Versión actualizada según nuevos contenidos

Tabla de contenido

Índice de Ilustraciones

Índice de Tablas

1. Agradecimientos

Vayan nuestros más sinceros agradecimientos a todos los docentes que durante estos 4 años nos prepararon para llegar hasta esta etapa y en especial al profesor Cristian Espinoza, quien nos entregó orientación, retroalimentación y apoyo durante todo el desarrollo del proyecto.

A los propietarios del Food Trucks “Cafeina Spa.” y en especial a “Erika Zapata” por permitirnos conocer sus operaciones, levantar los requerimientos y validar nuestra implementación inicial.

A la institución Duoc UC, por proporcionarnos acceso a recursos tecnológicos, licencias estudiantiles y el ambiente propicio para desarrollar nuestras competencias profesionales.

Y finalmente a nuestras familias, por su comprensión y apoyo incondicional durante las largas jornadas de trabajo y desarrollo.

2. Resumen Ejecutivo en Español

Según la Cámara Nacional de Comercio, Servicios y Turismo (CNC) y la Asociación Chilena de Gastronomía (ACHIGA), las ventas de comida rápida crecieron un 3,9% anual durante el último trimestre, acumulando un alza del 3% entre enero y septiembre del año 2025 (Cámara Nacional de Comercio, 2025) consolidándose como una alternativa gastronómica atractiva y accesible, sin embargo, en el ámbito de los "Food Truck" muchos de estos negocios operan con sistemas manuales, planillas Excel o soluciones genéricas que no contemplan las particularidades de la venta móvil, como la conectividad intermitente y la necesidad de rapidez en la atención.

Este proyecto Capstone desarrolla una Aplicación Web Progresiva (PWA) para la gestión integral de Food Trucks, enfocada en el procesamiento de pedidos, impresión automática de boletas y comandas, funcionalidad offline con sincronización automática, control de caja, historial de pedidos, integración con pasarela de pagos y soporte multirol/multisucursal. Inicialmente orientado a negocios de café, el sistema es modular y escalable para diferentes tipos de Food Trucks. Desarrollado con React en frontend, Django en backend, Azure SQL Server en base de datos y Azure DevOps para control de tareas, utiliza una metodología híbrida que combina planificación tradicional con Scrum ágil y Kanban para un desarrollo iterativo, garantizando eficiencia, seguridad y cumplimiento de normativas chilenas.

3. Executive Summary (Abstract)

According to the National Chamber of Commerce, Services and Tourism (CNC) and the Chilean Gastronomy Association (ACHIGA), during the first quarter of 2025, fast food sales grew by 3.9% annually during the last quarter, accumulating an increase of 3% between January and September of 2025 establishing itself as an attractive and accessible gastronomic alternative. However, in the Food Truck sector, many of these businesses operate with manual systems, Excel spreadsheets, or generic solutions that do not address the particularities of mobile sales, such as intermittent connectivity and the need for speed in service.

This Capstone project develops a Progressive Web Application (PWA) for comprehensive Food Truck management, focused on order processing, automatic printing of receipts and orders, offline functionality with automatic synchronization, cash control, order history, integration with a payment gateway, and multi-role/multi-branch support. Initially oriented toward coffee businesses, the system is modular and scalable for different types of Food Trucks. Built using React for frontend, Django for backend, Azure SQL Server for database, and Azure DevOps for task management, it employs a hybrid methodology that combines traditional planning with agile Scrum and Kanban for iterative development, ensuring efficiency, security, and compliance with Chilean regulations.

4. Introducción

La industria gastronómica móvil en Chile ha evolucionado significativamente y los Food Trucks han dejado de ser una alternativa ocasional para convertirse en emprendimientos formales que atienden eventos corporativos, ferias, festivales y ubicaciones fijas con alto tráfico peatonal. Esta transformación ha traído consigo nuevas exigencias: mayor profesionalización, cumplimiento tributario y financiero.

El contexto del proyecto se enmarca en esta realidad, los emprendedores de Food Trucks enfrentan desafíos como: operar en ambientes móviles con conectividad variable, requieren agilidad en la toma de pedidos para maximizar ventas en horarios peak, necesitan control de caja y deben cumplir con normativas del SII para emisión de boletas electrónicas.

Este proyecto nace de la identificación de estos desafíos y a través de entrevistas con propietarios de Food Trucks de café en Santiago, se validó la necesidad de una herramienta digital especializada, accesible económicamente, fácil de usar en condiciones móviles y que garantice continuidad operativa independiente de la conectividad.

El sistema desarrollado busca profesionalizar la operación de Food Trucks mediante tecnología apropiada, contribuyendo a la sostenibilidad y crecimiento de estos emprendimientos en el mercado chileno.

El informe a continuación detalla el proceso completo de diseño, desarrollo e implementación de esta solución, desde la conceptualización del problema hasta la validación de un producto funcional.

5. Descripción del Problema o Necesidad del Proyecto

El principal problema que aborda este proyecto es la falta de sistemas digitales especializados para la gestión operativa de Food Trucks en Chile. Durante entrevistas con emprendedores de cafeterías móviles en Santiago, se identificaron las siguientes problemáticas recurrentes:

- Gestión manual de pedidos: Los pedidos se anotan en papel, lo que aumenta el riesgo de errores, especialmente en horas punta.
- Falta de conectividad estable: Los Food Trucks operan en ferias, parques y eventos donde la señal de internet es limitada o inexistente, imposibilitando el uso de sistemas en la nube.
- Ineficiencia en la impresión: No existe automatización en la generación de comandas para cocina ni boletas electrónicas para clientes.
- Control de caja deficiente: El cierre diario se realiza manualmente, sin reportes automáticos ni conciliación con medios de pago.
- Ausencia de historial de ventas: No se lleva registro estructurado de pedidos, lo que dificulta el análisis de tendencias y toma de decisiones.

Por ello, existe una necesidad clara de una solución accesible y específica para el modelo de negocio de Food Trucks, que permita digitalizar sus operaciones sin complejidad ni altos costos.

6. Solución al Problema

La solución propuesta es un Sistema de Gestión Integral para Food Trucks basado en una Aplicación Web Progresiva (PWA), desarrollado bajo una arquitectura cliente-servidor de tres capas. Esta elección tecnológica permite acceso desde cualquier dispositivo móvil o tablet, sin necesidad de instalación desde tiendas de aplicaciones, y con funcionalidad completa incluso sin conexión a internet.

El sistema se implementó utilizando las siguientes tecnologías:

- Frontend: React.js con PWA Toolkit, para una interfaz responsiva.
- Backend: Django para exponer APIs seguras y escalables.
- Base de datos: Azure SQL Server.
- Almacenamiento local: IndexedDB y Service Workers para persistir datos en modo offline.
- Impresión: Integración con impresoras térmicas vía Bluetooth/USB mediante librerías JavaScript (escpos-thermal-print).
- Autenticación: JWT (JSON Web Tokens) con cifrado ARGON2.
- Gestión de tareas: Azure DevOps para seguimiento de sprints, backlog.

Desde el punto de vista humano, el proyecto fue desarrollado por un equipo de tres estudiantes de Ingeniería en Informática, con roles definidos.

En cuanto a la funcionalidad de pago se utilizará inicialmente la integración con Transbank utilizando su pasarela de pagos Web redirigiendo el proceso de pago a dicha plataforma, pero en un futuro cercano se espera incorporar una capa de integración con terminales POS físicas mediante APIs proporcionadas por Transbank, permitiendo la autorización automática de pagos, lo anterior es debido a que no contamos con acceso a terminales POS reales para realizar configuraciones ni pruebas.

Financieramente, el proyecto se ejecutó con un presupuesto reducido, aprovechando licencias educativas de Microsoft Azure y herramientas open source. Esto lo hace replicable y escalable para futuros emprendimientos.

7. Objetivos del Proyecto

7.1. Objetivo General

Diseñar e implementar un sistema digital integral para Food Trucks que optimice la gestión de pedidos, inventario, caja y ventas, garantizando operatividad en línea y fuera de línea, escalabilidad multi-sucursal y una experiencia de usuario eficiente y segura.

7.2. Objetivos Específicos

- Facilitar la captura rápida y precisa de pedidos con opciones de personalización según tipo de producto.
- Garantizar la funcionalidad offline con almacenamiento local y sincronización automática de datos.
- Integrar el sistema con la pasarela de pago de Transbank, permitiendo transacciones seguras.
- Controlar la apertura, cierre y movimientos de caja.
- Automatizar la generación e impresión de comandas para preparación y boletas para clientes.
- Implementar un historial de pedidos con capacidad para consultas y reportes.

8. Competencias del Perfil de Egreso

Este proyecto permite aplicar y demostrar las siguientes competencias del perfil de egreso:

- **Gestionar proyectos informáticos, ofreciendo alternativas para la toma de decisiones de acuerdo a los requerimientos de la organización**: Se aplica una metodología híbrida (tradicional + ágil con Kanban), permitiendo la planificación macro y ajustes basados en requerimientos funcionales y no funcionales.
- **Construir modelos de datos para soportar los requerimientos de la organización de acuerdo a un diseño definido y escalable en el tiempo**: El uso de Azure SQL Server para el modelado de bases de datos y poder manejar los pedidos, productos, boletas, etc.
- **Construir programas y rutinas de variada complejidad para dar solución a requerimientos de la organización, acordes a tecnologías de mercado y utilizando buenas prácticas de codificación**: El desarrollo modular con React facilitó el mantenimiento y evolución del sistema.
- **Realizar pruebas de certificación tanto de los productos como de los procesos utilizando buenas prácticas definidas por la industria**: Ejecutado mediante pruebas funcionales y de usabilidad.

9. Acta de Constitución del Proyecto

A continuación se detalla un resumen del acta de constitución del proyecto:

<table>
<tr><th>ITEM</th><th colspan="2">DESCRIPCIÓN</th></tr>
<tr><td>Nombre del proyecto</td><td colspan="2">Sistema de Gestión para Food Truck – Solución PWA</td></tr>
<tr><td>Breve descripción</td><td colspan="2">Desarrollo de una aplicación web progresiva para la gestión de pedidos de Food Trucks, con soporte offline, control de caja y pagos.</td></tr>
<tr><td>Objetivo general</td><td colspan="2">Diseñar e implementar un sistema digital integral para Food Trucks que optimice la gestión de pedidos, caja y ventas, garantizando operatividad online/offline y escalabilidad.</td></tr>
<tr><td>Objetivos específicos</td><td colspan="2">• Facilitar la captura rápida y precisa de pedidos con opciones de personalización según tipo de producto.
• Automatizar la generación e impresión de comandas para preparación y boletas para clientes.
• Implementar un historial de pedidos con capacidad para consultas y reportes.
• Garantizar la funcionalidad offline con almacenamiento local y sincronización automática de datos.
• Controlar la apertura, cierre y movimientos de caja con reportes financieros confiables.
• Integrar la plataforma con terminales de pago físicos, permitiendo transacciones automáticas y seguras.</td></tr>
<tr><td>Factores de éxito</td><td colspan="2">• MVP funcional entregado en plazo.
• Validación positiva por usuarios piloto.
• Cumplimiento de requisitos técnicos y legales.
• Documentación completa y código mantenible.</td></tr>
<tr><td rowspan="4">Fases del proyecto y entregables</td><td>Fase</td><td>Entregable</td></tr>
<tr><td>1. Inicio y Planificación</td><td>Requerimientos y planificación inicial.</td></tr>
<tr><td>2. Análisis y Diseño</td><td>Modelo entidad relación, diccionario de datos y diagramas 4+1.</td></tr>
<tr><td>3. Desarrollo</td><td>Aplicación funcional, Código fuente versionado en Git.</td></tr>
<tr><td></td><td>4. Pruebas y validaciones</td><td>Plan y casos de pruebas.</td></tr>
<tr><td></td><td>5. Cierre</td><td>Informe final, Brochure y Presentación.</td></tr>
<tr><td>Interesados clave</td><td colspan="2">• Equipo de desarrollo y gestión.
• Docente guía (Cristian Espinoza).
• Emprendedores de Food Trucks (usuario piloto)
• Escuela de Informática.</td></tr>
<tr><td>Riesgos</td><td colspan="2">• Dificultad técnica en sincronización offline.
• Uso total de créditos o finalización de licencias estudiantiles en Azure.</td></tr>
</table>

ITEM	DESCRIPCIÓN
Hitos principales	• Finalización Fase de Inicio y Planificación. • Finalización Fase de Análisis y Diseño. • Finalización Fase de Desarrollo. • Finalización Fase de Pruebas. • Finalización Fase de Cierre.
Presupuesto	$ 1.354.666 CLP
Requerimientos de aprobación	• Aprobación docente por fases. • Entrega sistema funcional. • Presentación ante la comisión evaluadora.
Líder del proyecto	René Veloso Salazar

Tabla 1 - Acta de Constitución del Proyecto

10.Asignación de Roles

NOMBRE	ROL	FUNCIONES Y TAREAS
René Veloso Salazar	Líder de Proyecto & Arquitecto de Datos	• Gestión general del proyecto • Diseño y modelado de la base de datos • Coordinación de tareas en Azure DevOps (tablero Kanban) • Control de calidad y definición de casos de prueba • Documentación técnica (MER, diccionario de datos, Informe final y presentación)
Vicente Gálvez Roldán	Lider Backend & Infraestructura	• Diseño infraestructura en Azure • Desarrollo de APIs en Django • Implementación de lógica de negocio • Autenticación y autorización (JWT) • Integración pasarela de pagos Transbank • Lógica de sincronización (resolución de conflictos)
Gustavo Pezzini Puen	Lider Frontend & UI/UX	• Desarrollo completo del frontend en React • Implementación de PWA (Service Workers, manifest) • Diseño de interfaz de usuario (UI) y experiencia de usuario (UX) • Integración de storage offline (IndexedDB)

Tabla 2 - Asignación de Roles

11.Metodología de Trabajo

Se adoptó una metodología híbrida que combina elementos de gestión tradicional con enfoques ágiles (Scrum y Kanban), que es apropiado para proyectos con alcance definido pero con un cierto grado de incertidumbre técnica.

FASE	DESCRIPCIÓN Y FUNCIÓN PRINCIPAL
Inicio y Planificación	Identificación de necesidades del sector, entrevistas a potenciales usuarios, definición del alcance y objetivos. Elaboración del plan macro del proyecto, definición de hitos, asignación de roles, elaboración de la Carta Gantt y acta de constitución del proyecto.
Análisis y Diseño	Análisis de requerimientos funcionales y no funcionales, diseño de la arquitectura del sistema (3 capas), modelado de datos (MER), diseño técnico de la solución (diagramas 4+1).
Desarrollo	Diseño, codificación e implementación iterativa e incremental del sistema. Esta fase se ejecutó mediante Scrum para el desarrollo del producto y Kanban transversal para el control visual de todas las tareas del proyecto. Cada sprint finalizaba con una revisión funcional y pruebas parciales.
Pruebas y validaciones	Ejecución de pruebas unitarias, de integración y de aceptación. Corrección de defectos y validación final del MVP.
Cierre	Documentación final,

Tabla 3 - Fases del Proyecto

12. Carta Gantt

La siguiente es la Carta Gantt del proyecto:

Nombre de tarea	Duración	% completado	Comienzo	Fin	Predec	Nombres de los recursos
Proyecto Sistema FoodTruck	**80,5 días**	**99%**	**lun 18/08/25**	**vie 05/12/25**		
Fase 1: Inicio y Planificación	**6 días**	**100%**	**lun 18/08/25**	**lun 25/08/25**		
Levantamiento de requerimientos y planificación macro	**6 días**	**100%**	**lun 18/08/25**	**lun 25/08/25**		
Recopilar requerimientos funcionales y no funcionales	2 días	100%	lun 18/08/25	mar 19/08/25		Gustavo Pezzini[25%]
Definir backlog inicial en Azure DevOps	2 días	100%	mié 20/08/25	jue 21/08/25	4	René Veloso[25%]
Planificación macro del proyecto	2 días	100%	vie 22/08/25	lun 25/08/25	5	René Veloso[25%]
Hito 1: Fin Levantamiento y Requerimientos	0 días	100%	vie 29/08/25	vie 29/08/25	6	
Fase 2: Análisis y Diseño	**9 días**	**100%**	**sáb 30/08/25**	**mié 10/09/25**		
Análisis y diseño de base de datos + Diagramas	**9 días**	**100%**	**sáb 30/08/25**	**mié 10/09/25**		
Análisis de entidades	2 días	100%	sáb 30/08/25	lun 01/09/25		René Veloso[25%]
Diseño de diagramas ER y UML	4 días	100%	mar 02/09/25	vie 05/09/25	10	René Veloso[25%]
Configuración inicial de Azure SQL Server	3 días	100%	sáb 30/08/25	mié 03/09/25		Vicente Galvez[25%];René Veloso[25%]
Hito 2: Fin Análisis y Diseño	0 días	100%	jue 04/09/25	jue 04/09/25		
Fase 3: Desarrollo (Ágil)	**24,5 días**	**100%**	**vie 05/09/25**	**jue 09/10/25**		
SPRINT 1: Infraestructura & Backend Core	**24,5 días**	**100%**	**vie 05/09/25**	**jue 09/10/25**		
Configuración de Entornos y Arquitectura Inicial	**13 días**	**100%**	**vie 05/09/25**	**mar 23/09/25**		
Setup de entorno Django y Azure	4,5 días	100%	vie 05/09/25	jue 11/09/25		Vicente Galvez[25%]
Desarrollo de modelos de datos	6 días	100%	jue 11/09/25	vie 19/09/25		René Veloso[25%]
Instalación prerrequisitos, vs code, node.js	1,5 días	100%	vie 19/09/25	lun 22/09/25		Gustavo Pezzini[25%]
Instalación vite, react, tailwindcss, dependencias	1 día	100%	mar 23/09/25	jue 25/09/25		Gustavo Pezzini[25%]
Configurar vs code y extensiones	0,88 días	100%	vie 26/09/25	vie 26/09/25		Gustavo Pezzini[25%]
Implementación de APIs básicas para Login usuarios	5 días	100%	dom 28/09/25	vie 03/10/25		Vicente Galvez[25%]
CRUD gestión de pedidos	5 días	100%	sáb 04/10/25	vie 10/10/25		Vicente Galvez[25%]
SPRINT 2: Frontend Core & PWA	**13,6 días**	**100%**	**mar 23/09/25**	**vie 10/10/25**		
Arquitectura Frontend y Enrutamiento	6 días	100%	mar 23/09/25	mar 30/09/25		Gustavo Pezzini[25%]
Desarrollo Módulo de Autenticación	1,5 días	100%	vie 03/10/25	lun 06/10/25	25	Gustavo Pezzini[25%];Vicente Galvez[25
Desarrollo Dashboard Vendedor y Catálogo	2 días	100%	vie 10/10/25	lun 13/10/25	26	Gustavo Pezzini[25%];Vicente Galvez[25
Desarrollo Módulo Toma de Pedidos	1,85 días	100%	jue 09/10/25	vie 10/10/25	27	Gustavo Pezzini[25%];Vicente Galvez[25
Pruebas validaciones	16 horas	100%	lun 20/10/25	mar 21/10/25	28	René Veloso[25%]
SPRINT 3: Integración & Sincronización	**18,7 días**	**100%**	**vie 24/10/25**	**mar 18/11/25**		
Setup de PWA manifest	1 día	100%	vie 24/10/25	vie 24/10/25		Gustavo Pezzini[25%]
Implementación de modo offline	2 días	100%	lun 27/10/25	mar 28/10/25	31	Gustavo Pezzini[25%]
Integración frontend-backend vía APIs	2,69 días	100%	mié 29/10/25	vie 31/10/25	32	Gustavo Pezzini[25%];Vicente Galvez[2
Desarrollo de sincronización offline-online	4 días	100%	sáb 01/11/25	jue 06/11/25	33	Gustavo Pezzini[25%]
Integración con POS simulado	5 días	100%	vie 07/11/25	jue 13/11/25	34	Gustavo Pezzini[25%];Vicente Galvez[2
Manejo de roles multiusuario	2,7 días	100%	vie 14/11/25	mar 18/11/25	35	Gustavo Pezzini[33%];Vicente Galvez[3
Pruebas y validaciones	0 días	100%	lun 17/11/25	lun 17/11/25		René Veloso
SPRINT 4: Funcionalidades Avanzadas & Optimización	**5 días**	**100%**	**mar 18/11/25**	**lun 24/11/25**		
Implementación de reportes financieros y historial	0,5 días	100%	mar 18/11/25	mar 18/11/25		Gustavo Pezzini;Vicente Galvez
Funcionalidad Apertura y Cierre de Caja	0,25 días	100%	jue 20/11/25	jue 20/11/25		Gustavo Pezzini;Vicente Galvez
Funcionalidades multisucursal	0,5 días	100%	vie 21/11/25	vie 21/11/25		Vicente Galvez;Gustavo Pezzini
Pruebas y validaciones	3 horas	100%	sáb 22/11/25	sáb 22/11/25		René Veloso
Hito 3: Fin Desarrollo	0 días	100%	sáb 22/11/25	sáb 22/11/25		
Fase 4: Pruebas y Validaciones	**2 días**	**100%**	**jue 20/11/25**	**vie 21/11/25**		
UAT y Despliegue	**2 días**	**100%**	**jue 20/11/25**	**vie 21/11/25**		
Pruebas de Aceptación (UAT)	2 días	100%	jue 20/11/25	vie 21/11/25		René Veloso[25%]
Despliegue en Azure App Service y Static Web Apps	1 día	100%	vie 21/11/25	vie 21/11/25		Vicente Galvez
Hito 4: Fin Pruebas	0 días	100%	vie 21/11/25	vie 21/11/25	47	
Fase 5: Cierre	**15,5 días**	**92%**	**sáb 15/11/25**	**vie 05/12/25**		
Documentación Final	**10,9 días**	**96%**	**sáb 15/11/25**	**vie 28/11/25**		
Informe de proyecto	10 días	100%	sáb 15/11/25	jue 27/11/25		René Veloso
Preparación video de demo funcional	0,5 días	0%	mar 25/11/25	mar 25/11/25		Gustavo Pezzini
Presentación	1 día	100%	mié 26/11/25	vie 28/11/25	51	René Veloso
Entrega y Presentación	**1,5 días**	**0%**	**mié 03/12/25**	**jue 04/12/25**		
Presentación ante comisión	0,5 días	0%	vie 05/12/25	vie 05/12/25		Gustavo Pezzini;René Veloso;Vicente G
Hito 5: Fin Cierre	0 días	0%	vie 05/12/25	vie 05/12/25		

Ilustración 1 - Gantt del Proyecto

13.Marco Teórico o Conceptual

13.1. Fundamentos Tecnológicos

A. Arquitectura de Software Cliente-Servidor de Tres Capas

El sistema se basa en una arquitectura cliente-servidor tradicional de 3 capas, que separa responsabilidades y facilita escalabilidad y mantenimiento.

Capa de Presentación (Frontend):

- Responsabilidad: Interfaz de usuario, interacción, visualización
- Tecnología: React (biblioteca JavaScript), PWA
- Características: Responsiva, táctil, almacenamiento local (IndexedDB)

Capa de Lógica de Negocio (Backend):

- Responsabilidad: Procesamiento de reglas de negocio, autenticación, APIs
- Tecnología: Django (Python)
- Características: API RESTful, validaciones, lógica de sincronización

Capa de Datos (Base de Datos):

- Responsabilidad: Persistencia, consultas, integridad referencial
- Tecnología: Azure SQL Server (Microsoft)
- Características: Relacional, transaccional, escalable

Beneficios de esta arquitectura:

- Separación de responsabilidades
- Facilita testing independiente de cada capa
- Permite escalar horizontalmente el backend
- Cambios en una capa no afectan necesariamente a las otras

B. Progressive Web Applications (PWA)

Las PWA combinan lo mejor de aplicaciones web y nativas, siendo clave para contextos móviles como Food Trucks.

Características principales:

- Instalables: Se añaden a la pantalla de inicio sin tiendas de aplicaciones
- Offline-first: Funcionan sin conexión mediante Service Workers
- Seguras: Requieren HTTPS para habilitar funciones avanzadas
- Responsivas: Se adaptan a cualquier tamaño de pantalla

Tecnologías PWA implementadas:

- Service Workers: Scripts que interceptan peticiones de red, permiten cache y sincronización en background
- Manifest.json: Archivo de configuración (nombre app, íconos, colores, orientación)
- IndexedDB: Base de datos local del navegador para almacenamiento estructurado offline

Justificación para Food Trucks:

- No requiere instalación desde tiendas (distribución inmediata)
- Funciona en tablets Android/iOS sin desarrollo nativo
- Operación continua aunque falle la conexión en ubicaciones remotas
- Menor consumo de recursos que apps nativas

C. API REST (Representational State Transfer)

El backend expone una API REST para comunicación con el frontend y potenciales integraciones futuras.

Principios REST aplicados:

1. **Recursos identificados por URI:** /api/v1/productos/{producto_id}
2. **Operaciones mediante verbos HTTP:**
    - GET (lectura)
    - POST (creación)
    - PUT/PATCH (actualización completa/parcial)
    - DELETE (eliminación)
3. **Stateless:** Cada petición contiene toda la información necesaria (token JWT)
4. **Representaciones JSON:** Formato estándar para intercambio de datos

Ventajas:

- Independencia entre frontend y backend
- Documentación clara con Swagger UI
- Facilita integración con sistemas externos (POS, contabilidad)
- Escalabilidad horizontal (añadir más servidores backend)

D. Autenticación y Autorización con JWT

JSON Web Tokens (JWT) es un estándar abierto para transmitir información de forma segura entre partes.

Flujo de autenticación:

1. Usuario envía credenciales (username, password) a /api/auth/login/

2. Backend valida y retorna JWT (access token + refresh token)
3. Frontend almacena tokens (memoria o localStorage)
4. Cada petición incluye: Authorization: Bearer <access_token>
5. Backend verifica firma del token, extrae user_id y role
6. Cuando access token expira (60 min), se usa refresh token para obtener uno nuevo.

E. Sincronización Offline-Online

La sincronización bidireccional es crítica para garantizar coherencia de datos.

Estrategia implementada:

Almacenamiento offline (IndexedDB):

- Réplica local de productos, categorías, configuraciones
- Cola de operaciones pendientes (pedidos, movimientos de caja)
- Timestamp en cada registro para control de versiones

Algoritmo de sincronización:

1. **Detección de conectividad:** Event listener online/offline + ping periódico al backend
2. **Envío al servidor (push):**
   - Iterar cola de operaciones locales pendientes
   - POST/PUT cada registro al backend
   - Marcar como sincronizado si respuesta exitosa (200/201)
   - Reintentar si falla
3. **Descarga desde servidor (pull):**
   - GET de actualizaciones: ?updated_after=<último_timestamp_local>
   - Merge con datos locales
4. **Resolución de conflictos:**
   - Last-Write-Wins: Se mantiene el registro con timestamp más reciente
   - Casos edge: Modificación simultánea de inventario se resuelve en backend con locks

13.2. Revisión Solución Similar (estado del arte)

La solución seleccionada es el "KIT Punto de Venta para Cafetería, Comida Rápida y FoodTruck" de la empresa POSoft C y F (POSOFT, 2025), que es una solución integral diseñada para optimizar la gestión de negocios del rubro gastronómico móvil y fijo. Su propuesta combina hardware reacondicionado (computador básico Intel i3, 4 GB RAM, Windows 10 Enterprise LTSC) con un software especializado de restaurant licenciado de por vida bajo versión MONOCAJA.

Características principales:

- Control total de operaciones con gestión de inventarios, stock y ventas.
- Permite organización por mesas y camareros, agilizando la atención y el flujo en cocina a través de impresión de comandas.
- El kit incluye impresora térmica para tickets de caja y comandas (tamaño 80 mm) y gaveta de dinero con apertura electrónica.
- El software agiliza el servicio sin necesidad de códigos de barra y facilita el valor instantáneo de productos.
- La garantía varía entre 2 meses para productos reacondicionados y 6 meses para productos nuevos.
- Ideal para food trucks, cafeterías y locales de comida rápida, con un enfoque en mejorar la eficiencia y control dentro del negocio.

Precio

- El kit tiene un costo aproximado de $389.990 CLP (precio puede variar).
- Incluye todo lo necesario para iniciar operaciones.
- Es una solución pensada para negocios pequeños y medianos que buscan una opción asequible y práctica con soporte incluido.

14.Requerimientos del Sistema

14.1. Requerimientos Funcionales

Grupo Requerimientos	ID Requerimiento	Descripción Requerimiento
RF-001: Gestión de Pedidos y Ventas	RF-001-A	El sistema debe proporcionar una interfaz optimizada para la toma rápida de pedidos, con categorización de productos.
	RF-001-B	Debe permitir la modificación dinámica de cantidades, eliminación de ítems y aplicación de descuentos.
	RF-001-C	El cálculo de totales debe incluir impuestos (IVA) y redondeo según normativas vigentes.
	RF-001-D	Debe soportar múltiples métodos de pago (efectivo, tarjetas) con integración a terminales POS.
RF-002: Personalización y Configuración de Productos	RF-002-A	El sistema debe permitir la definición de modificadores dinámicos por producto (ingredientes, tamaños, extras).
	RF-002-B	Debe soportar reglas de negocio configurables (combinaciones no permitidas, productos obligatorios.
	RF-002-C	La configuración de productos debe ser centralizada pero personalizable por punto de venta.
RF-003: Gestión Documentos	RF-003-A	Generación automática de comandas de cocina con información clara de productos.
	RF-003-B	Emisión de boletas cumpliendo normativas del SII chileno.
	RF-003-C	Impresión automática en impresoras térmicas.
RF-004: Operación Offline y Sincronización	RF-004-A	El sistema debe mantener funcionalidad sin conexión a internet.
	RF-004-B	Sincronización automática e incremental y validación de integridad de datos.
	RF-004-C	Información del estado de conectividad.
RF-005: Control de Caja y Finanzas	RF-005-A	Apertura/cierre de caja con cuadre automático y registro de diferencias.
	RF-005-B	Seguimiento de ingresos por método de pago.

Grupo Requerimientos	ID Requerimiento	Descripción Requerimiento
RF-006: Gestión Multiusuario y Multisucursal	RF-006-A	Sistema de roles granular (Administrador, Supervisor, Cajero) con permisos específicos.
	RF-006-B	Gestión centralizada de múltiples puntos de venta con datos consolidados.
	RF-006-C	Auditoría completa de acciones por usuario.

Tabla 4 - Listado de Requerimientos Funcionales

14.2. Requerimientos No Funcionales

Grupo Requerimientos	ID Requerimiento	Descripción Requerimiento
RNF-001: Rendimiento y Escalabilidad	RNF-001-A	Tiempo de respuesta inferior a 10 segundos para operaciones críticas (toma de pedidos).
	RNF-001-B	Escalabilidad horizontal para soportar múltiples puntos de venta simultáneos.
RNF-002: Seguridad y Cumplimiento	RNF-002-A	Autenticación con cifrado end-to-end para datos sensibles.
	RNF-002-B	Cumplimiento con normativas del SII para boleta electrónica.
	RNF-002-C	Backup automático y recuperación ante desastres.
RNF-003: Usabilidad	RNF-003-A	Interfaz responsive optimizada.
	RNF-003-B	El sistema debe ofrecer una interfaz de usuario intuitiva, eficiente y adaptada al entorno operativo de un Food Truck.
RNF-004: Disponibilidad y Confiabilidad	RNF-004-A	Disponibilidad del 99.5% en modo online y 100% en modo offline.

Tabla 5 - Listado de Requerimientos no Funcionales

14.3. Matriz de Trazabilidad

ID Requerimiento	Modulo	Prueba Realizada
RF-001-A	Módulo de Pedidos	Verificar que el usuario pueda navegar entre categorías (café, bebidas, snacks) y seleccionar productos en menos de 5 segundos por ítem.
RF-001-B	Módulo de Pedidos	Simular un pedido en curso; validar que se pueden modificar cantidades, eliminar productos y aplicar descuentos sin errores.
RF-001-C	Módulo de Pagos	Validar que el IVA (19%) se aplica correctamente y que el redondeo cumple con la normativa chilena (múltiplo de $5).
RF-001-D	Módulo de Pagos	Conectar terminal POS simulada (Transbank); verificar transacción exitosa y registro correcto en el sistema.
RF-002-A	Módulo Mantenedores	Crear un producto con modificadores (ej. tamaño, leche vegetal); verificar que se aplican correctamente al pedido.
RF-002-B	Módulo Mantenedores	Configurar una regla (ej. “limón obligatorio en té helado”); probar escenarios válidos e inválidos.
RF-002-C	Módulo Mantenedores	Modificar un producto en una sucursal específica; verificar que cambios locales no afecten otras sucursales.
RF-003-A	Módulo de Pedidos	Emitir un pedido y verificar que la comanda impresa contiene hora, número, productos, modificadores y observaciones.
RF-003-B	Módulo de Pagos	Generar boleta electrónica con RUT, timbre SII y firma digital; validar contra esquema XML oficial.
RF-004-A	Módulo de Pedidos	Desconectar dispositivo; realizar pedidos y pagos; verificar que todas las funciones siguen disponibles.
RF-004-B	Módulo de Pedidos	Al restablecer conexión, verificar que los datos se sincronizan sin duplicados y con marca de tiempo.
RF-004-C	Interfaz de Usuario (UI)	Verificar que la UI muestra claramente: “Conectado”, “Modo Offline” o “Sincronizando...”.
RF-005-A	Módulo de Caja	Abrir caja con monto inicial; registrar ventas; cerrar caja y comparar ingresos reales vs. registrados.
RF-006-A	Módulo de Usuarios y Permisos	Asignar roles; verificar que el Cajero no puede acceder a configuración ni reportes financieros.
RF-006-B	Módulo de Sucursales	Registrar movimientos en dos sucursales; desde el panel central, visualizar ventas consolidadas.
RF-006-C	Módulo de Auditoría	Realizar acciones clave (cambiar precio, cerrar caja); verificar que quedan registradas con fecha y usuario.
RNF-001-A	Todos los módulos (frontend/backend)	Medir tiempo de carga de pantalla de pedidos y generación de boleta bajo condiciones normales (red 4G, dispositivo móvil).
RNF-001-B	Backend / Infraestructura	Simular 10 sucursales activas enviando datos simultáneamente; verificar que el servidor responde sin caídas.
RNF-002-A	Seguridad / Backend	Usar Burp Suite para verificar que contraseñas y tokens están cifrados en tránsito (HTTPS + JWT firmado).
RNF-002-B	Módulo de Pagos	Enviar boleta al ambiente de pruebas del SII; verificar aceptación y recepción de acuse de recibo.

ID Requerimiento	Modulo	Prueba Realizada
RNF-003-A	Frontend / UI	Ejecutar PWA en tablet; verificar diseño adaptado, botones táctiles y legibilidad.

Tabla 6 - Matriz de Trazabilidad

15. Diseño de Interfaz y Experiencia de Usuario (UX/UI)

El diseño de la interfaz y la experiencia de usuario ha sido desarrollado con un enfoque centrado en el usuario, priorizando la usabilidad, accesibilidad y coherencia visual considerando desde las primeras etapas del desarrollo los principios fundamentales de UX/UI, asegurando una experiencia fluida e intuitiva. Entre los principales componentes utilizados destacan los siguientes:

- **Layouts**: Estructuras base para rutas protegidas y públicas (por ejemplo, áreas solo para usuarios autenticados vs. accesos de invitados).
- **Tarjetas de Producto**: Muestran fotografía, nombre y precio de cada ítem para selección rápida.
- **Modales**: Ventanas emergentes para confirmaciones o formularios ágiles (como agregar modificadores).
- **Botones**: Clasificados según acción (Primario, Secundario, etc.), diseñados y estandarizados mediante utilidades de TailwindCSS.

15.1. Principios de Diseño Aplicados

Se han aplicado los siguientes principios para garantizar una Experiencia de Usuario (UX) óptima:

1. **Coherencia Visual (Design System):**
   - Se utiliza Tailwind CSS para mantener una consistencia en espaciados (paddings/margins), tipografía y paleta de colores en todas las pantallas. Esto reduce la carga cognitiva del usuario, ya que los elementos interactivos (botones, inputs) se comportan siempre de la misma manera.
2. **Enfoque "Offline-First" (UX de Conectividad):**
   - Dado que la aplicación opera en entornos con conectividad inestable, la interfaz no se bloquea cuando falta internet. Se implementa un diseño optimista donde las acciones del usuario se confirman instantáneamente en la interfaz mientras la persistencia de datos ocurre en segundo plano usando Dexie.js (IndexedDB).
3. **Diseño Responsive y Adaptativo:**

- La interfaz se adapta automáticamente a diferentes tamaños de pantalla, priorizando tablets para los vendedores, y pantallas de escritorio para los administradores.

4. **Accesibilidad:**
   - Uso de colores con alto contraste para legibilidad en exteriores (común en food trucks) y elementos táctiles de tamaño adecuado (min 44x44px) para evitar errores de pulsación.

15.2. Roles y Flujo Principal de Venta

La arquitectura de la información se divide claramente según los permisos del usuario. A continuación se detalla el flujo principal de venta, que es el núcleo del negocio.

Roles de Usuario

- **Administrador**: Acceso total a métricas, configuración y gestión de usuarios.
- **Supervisor**: Capacidad de autorizar operaciones especiales (anulaciones, descuentos altos) y supervisar turnos.
- **Vendedor**: Acceso restringido al flujo de ventas y cierre de caja.

Flujo Principal: Proceso de Venta (Vendedor)

Este fujo describe la interacción paso a paso del vendedor con el sistema para completar una transacción:

1. **Autenticación:**
   - El usuario ingresa sus credenciales, si son válidas ingresa al sistema.
2. **Selección de Productos:**
   - El vendedor selecciona ítems desde el catálogo visual o utiliza el buscador. Los productos se añaden al "Carrito Virtual" visible en pantalla.
3. **Gestión de la Orden:**
   - El sistema permite modificar cantidades o añadir modificadores (ej. "sin azúcar").
4. **Cobro y Pago:**
   - Se selecciona el método de pago (Efectivo, Tarjeta).
   - Al confirmar, la transacción se guarda inmediatamente en la base de datos local (IndexedDB).
5. **Sincronización (Post-Venta):**
   - El sistema intenta enviar la venta al servidor central.
   - Escenario Sin Conexión: Si no hay internet, la venta queda marcada como "Pendiente de Sync" (icono amarillo).
   - Escenario Con Conexión: La venta se sube y se marca como "Sincronizada" (icono verde).
6. **Emisión de Comprobante:**
   - Generación digital del ticket de venta.

15.3. Prototipo de Pantallas Representativas

1. **Login de Acceso:**

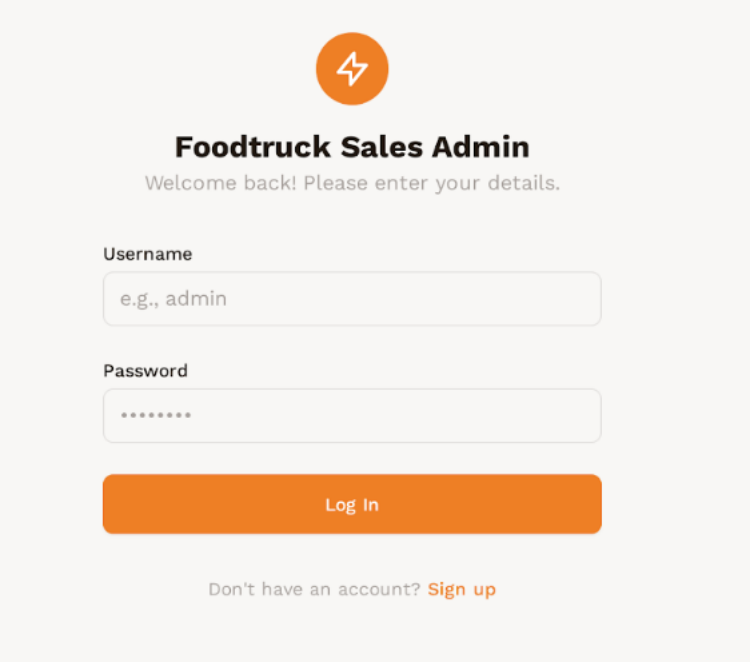

Ilustración 2 - Login de Acceso / Prototipo

2. **Gestión de Productos:**

Ilustración 3 - Gestión Productos / Prototipo

3. Gestión de Pedido y Carro de Compra:

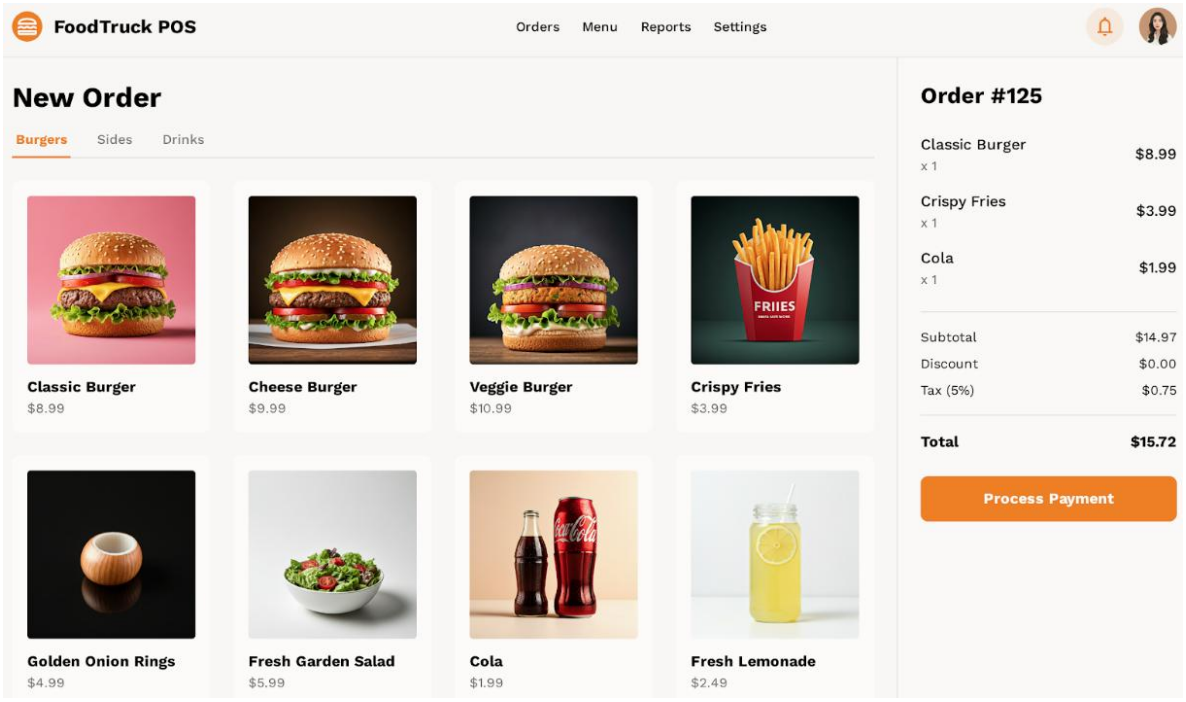

Ilustración 4 - Gestión Pedido / Prototipo

4. Dashboard Administrativo:

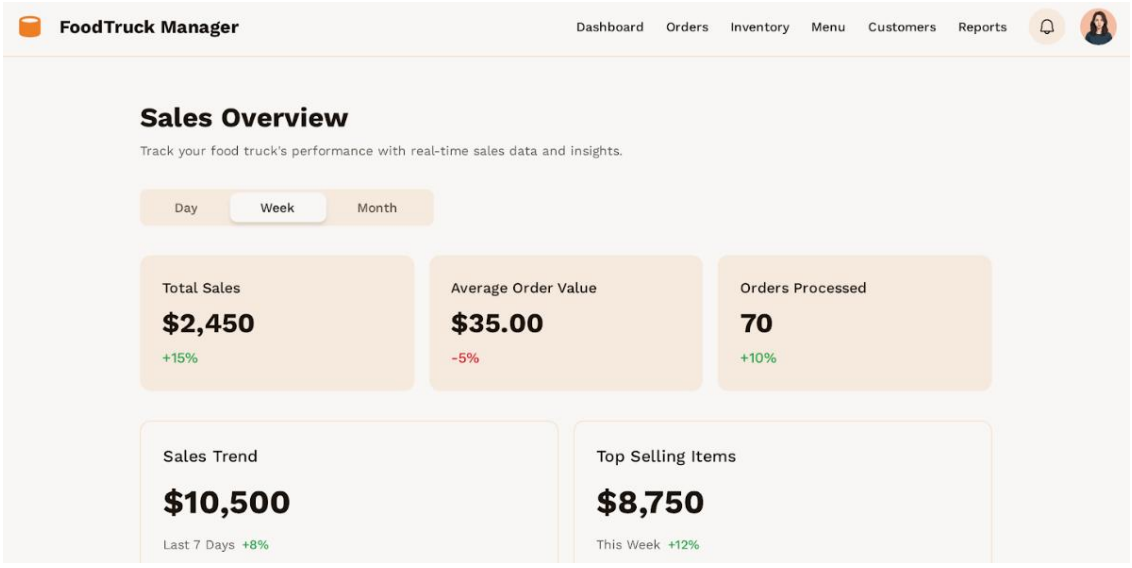

Ilustración 5 – Dashboard Administrativo / Prototipo

16. Implementación del Proyecto

16.1. Descripción General del Sistema

El Sistema de Gestión para Food Trucks es una solución digital que centraliza las operaciones de venta, control financiero y reportes para negocios gastronómicos móviles. El sistema está diseñado para operar de forma confiable tanto con conectividad a Internet como sin ella, garantizando continuidad operativa en cualquier ubicación.

Público objetivo:

- Propietarios de Food Trucks pequeños y medianos (1-10 unidades)
- Sector gastronómico: café móvil, comida rápida, helados, bebidas
- Emprendedores que buscan profesionalizar su operación

Módulos principales:

1. **Gestión de Pedidos:** Captura rápida, personalización de productos, carrito inteligente
2. **Control de Caja:** Apertura, movimientos, cierre con cuadre automático
3. **Impresión:** Comandas de cocina y boletas de cliente
4. **Sincronización:** Motor offline-online con resolución de conflictos
5. **Integración Transbank:** Pagos vía la pasarela Web en entorno Sandbox
6. **Reportes:** Dashboards con ventas por sucursal, productos más vendidos.
7. **Administración:** Usuarios, productos, sucursales, configuración

16.2. Modelado Técnico

A. Diagrama de Arquitectura General

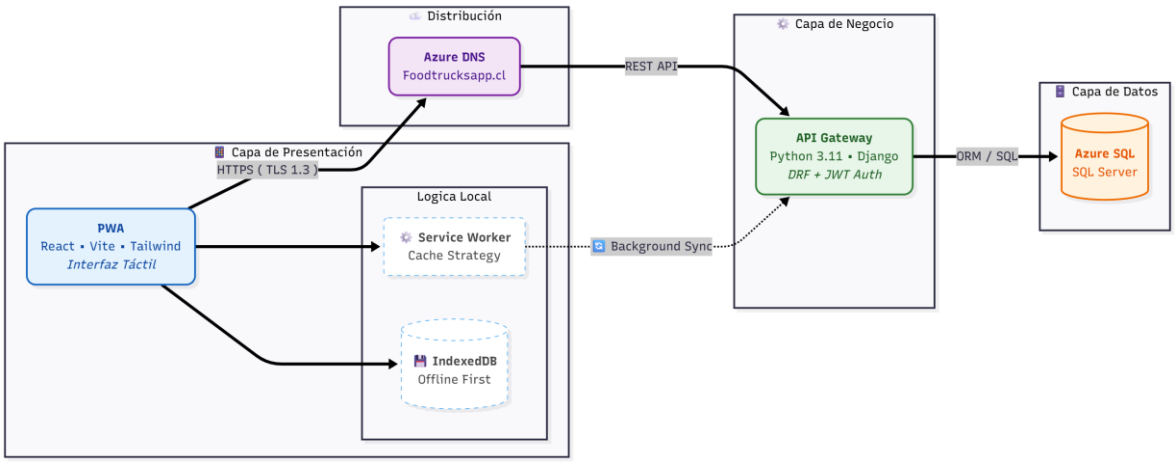

Ilustración 6 - Diagrama Arquitectura

B. Diagrama de Entidad Relación

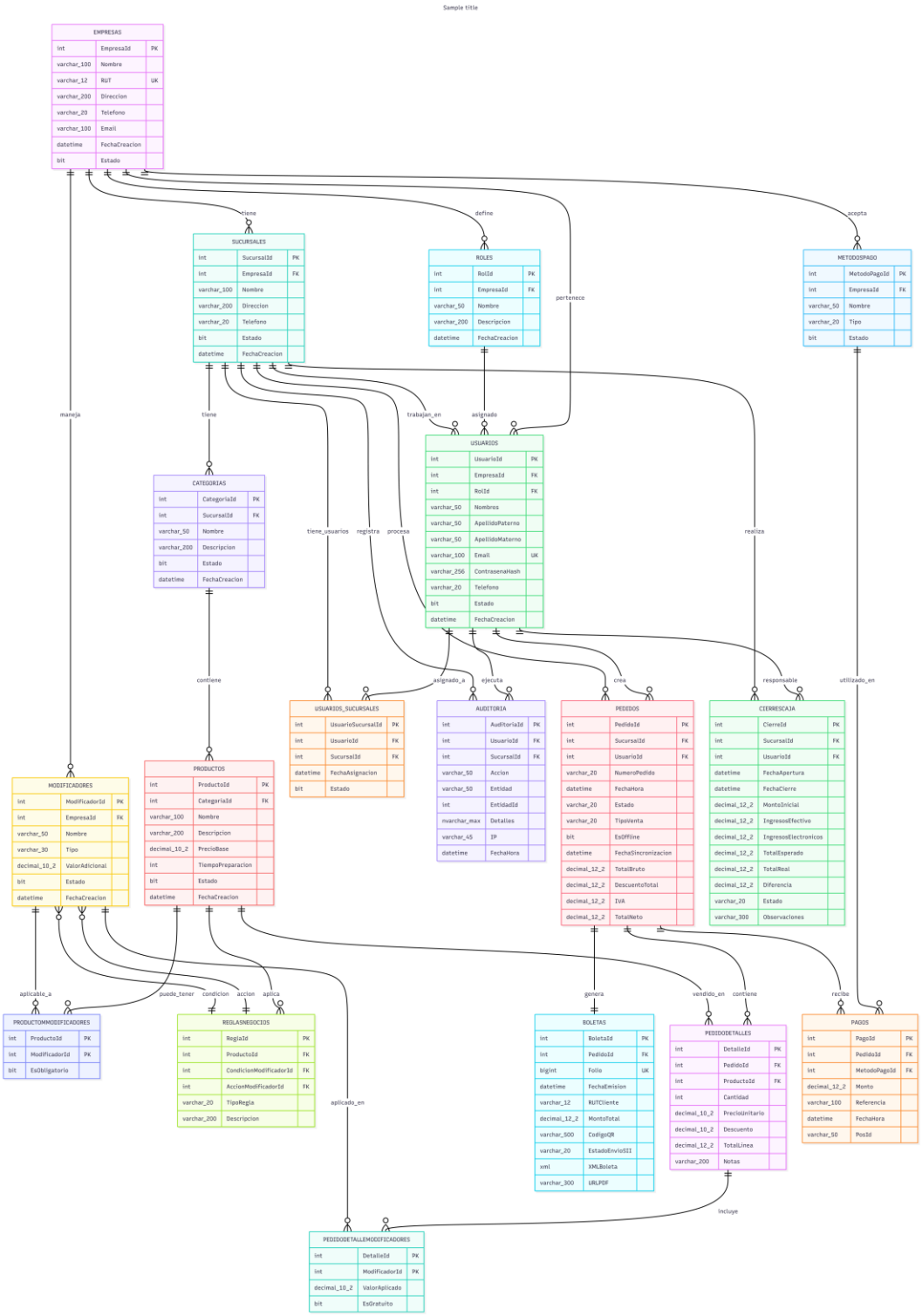

Ilustración 7 - Diagrama Entidad Relación

C. Diagramas 4 + 1

C.1. Vista Lógica - Diagrama de Clases

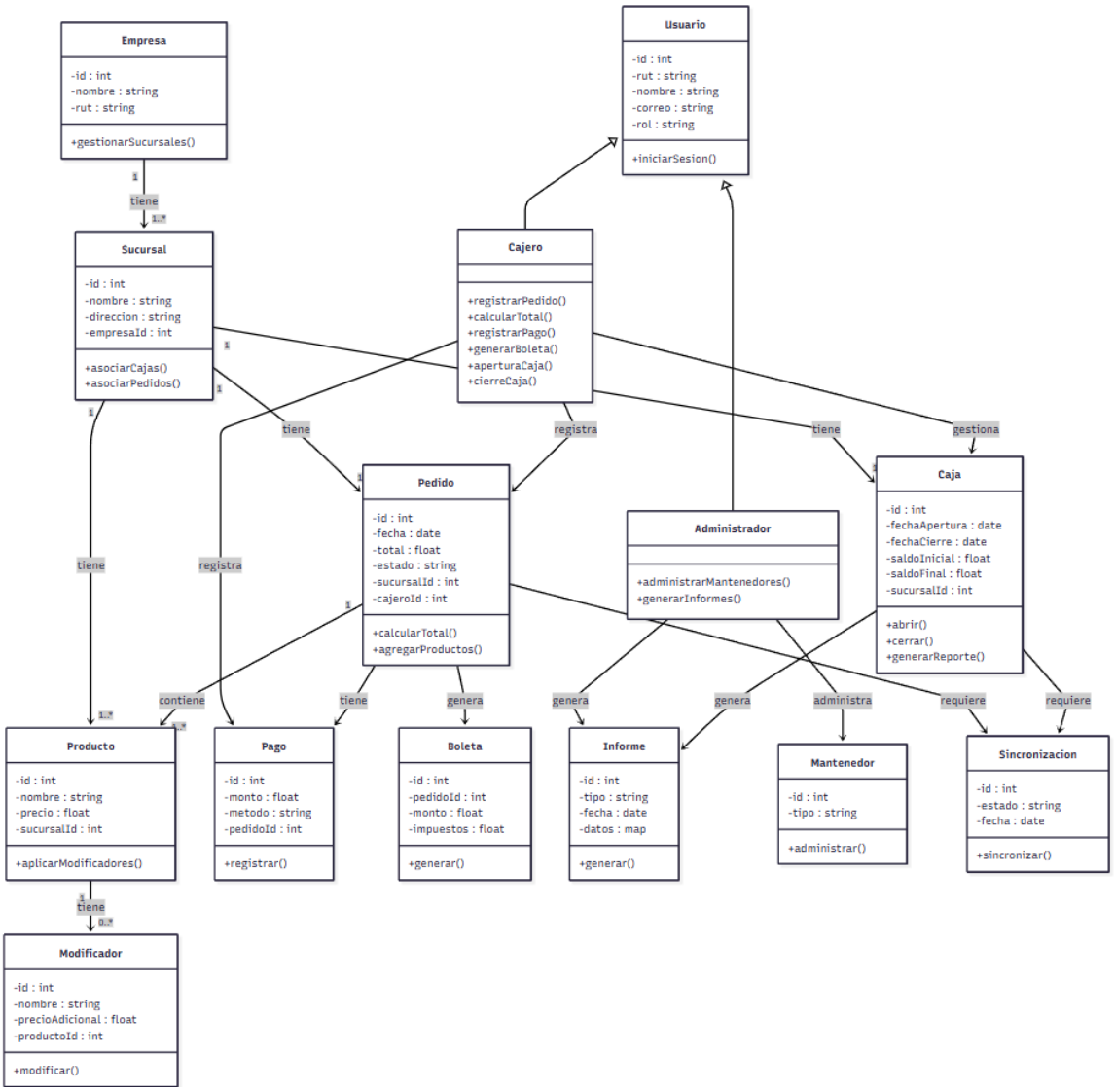

Ilustración 8 - Diagrama de Clases

C.2. Vista de procesos - Diagrama de Secuencia

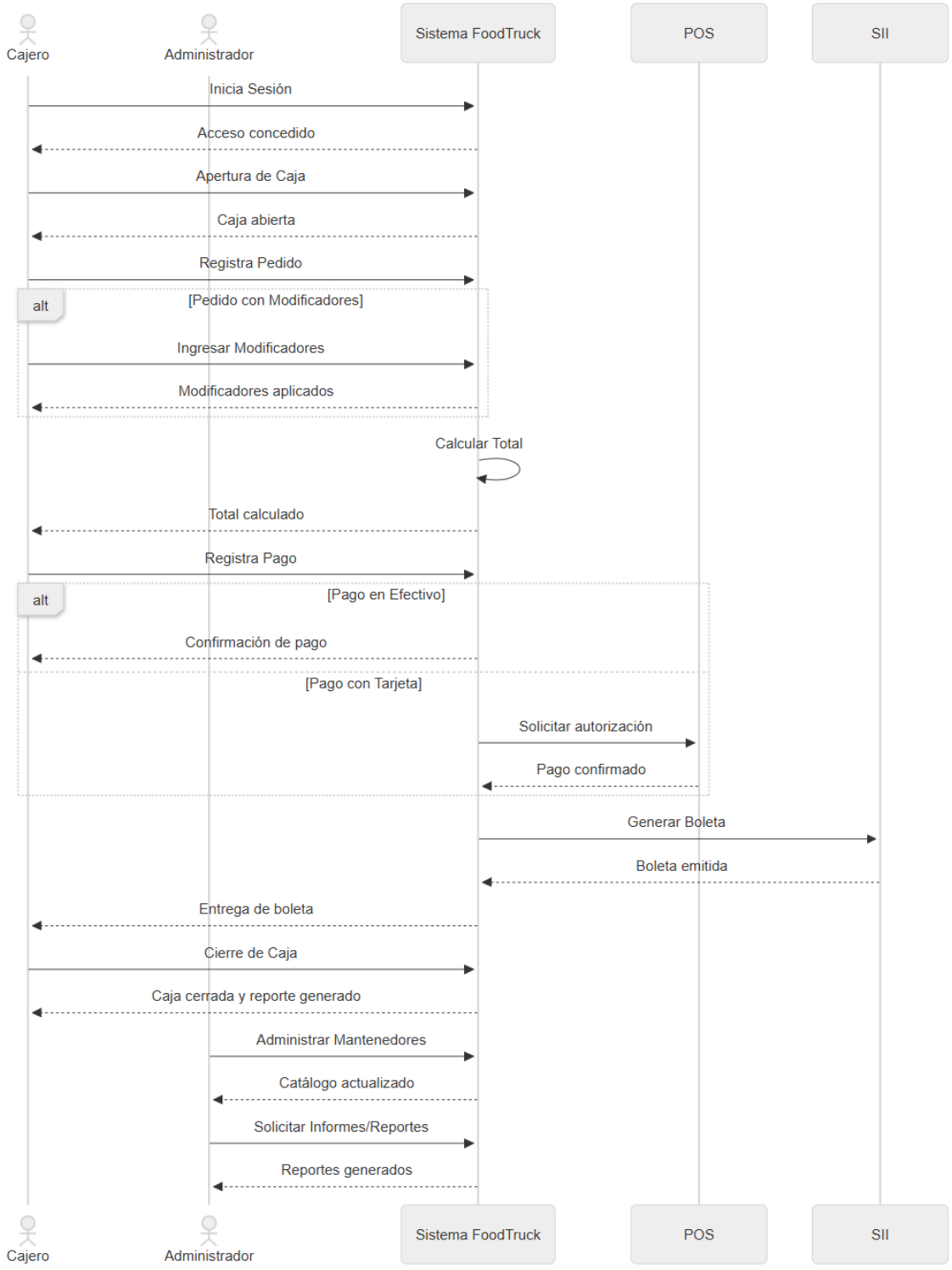

Ilustración 9 - Diagrama de Secuencia

C.3. Vista de Desarrollo (implementación) - Diagrama de Componentes

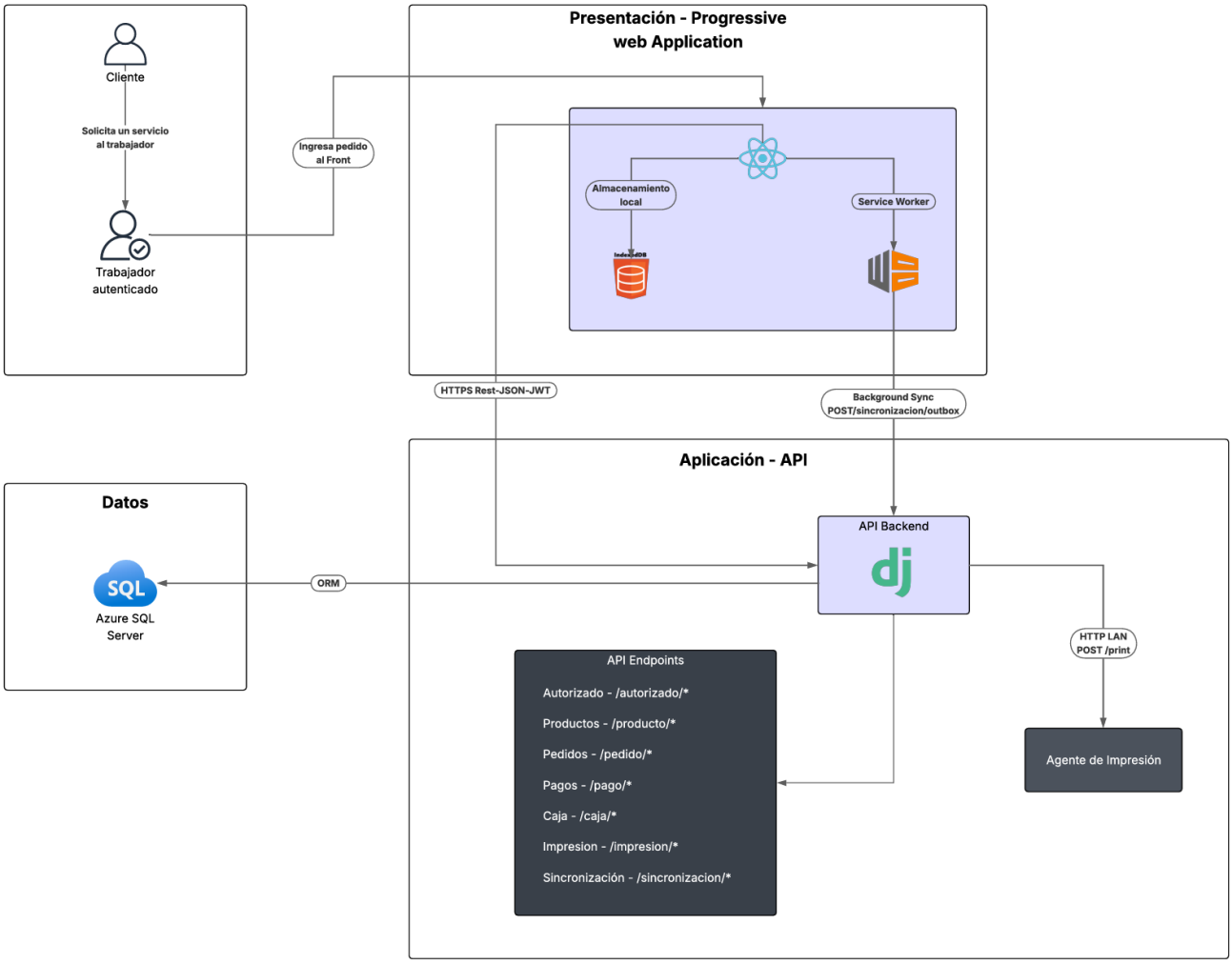

Ilustración 10 - Diagrama de Componentes

C.4. Vista Física (de despliegue) - Diagrama de Despliegue

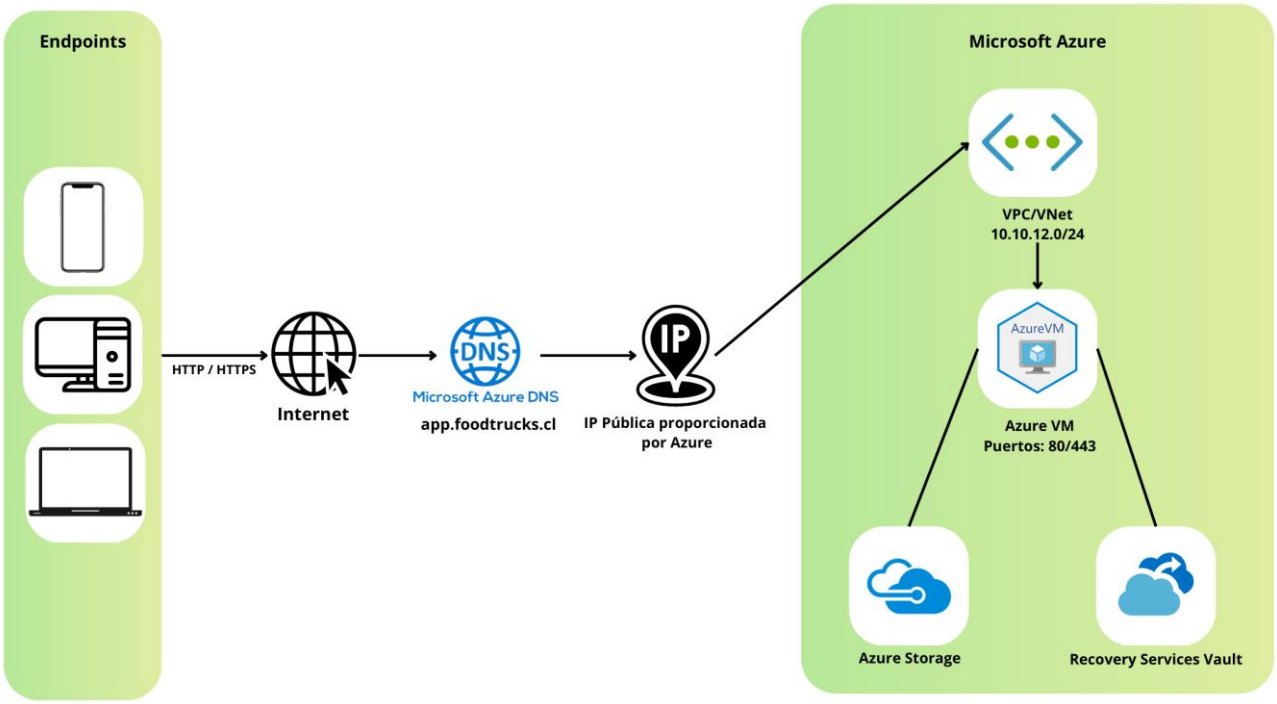

Ilustración 11 - Diagrama de Despliegue

C.5. Vista de Escenarios (+1) - Caso de Uso General

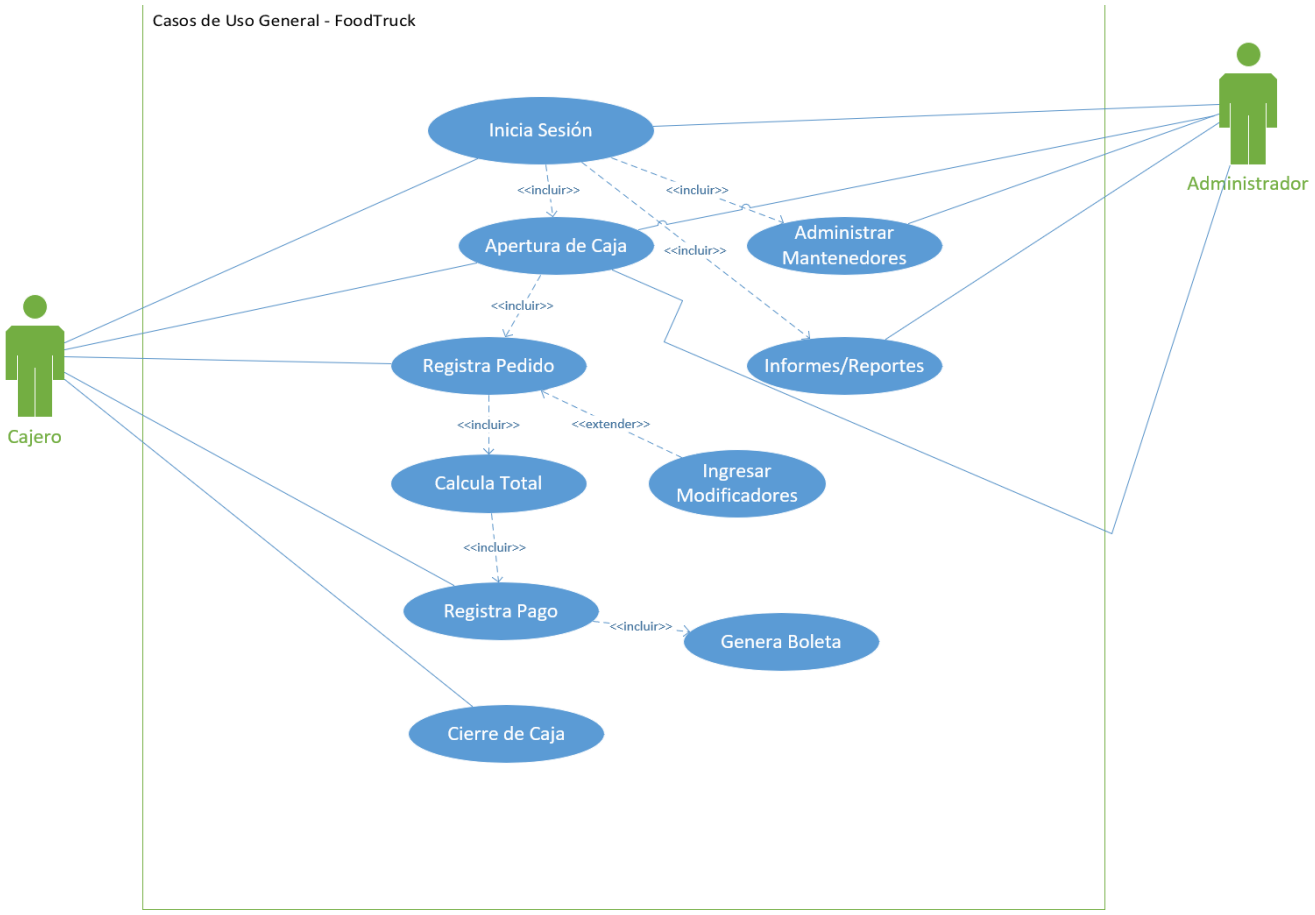

Ilustración 12 - Diagrama Caso de Uso

16.3. Decisiones Técnicas y Tecnologías Aplicadas

Tecnologías, Frameworks y Librerías

La selección tecnológica responde directamente a los requerimientos críticos de movilidad, desconexión (Offline) y bajo costo operativo.

- **Frontend: React.js (v18+)**
  - **Justificación**: Se eligió por su arquitectura basada en componentes, lo que facilita la creación de una interfaz modular (Botones de producto,

Ticket de venta, Panel de control). Además, su ecosistema es el más robusto para transformar una web en una **PWA (Progressive Web App)**.

- **PWA Core: Service Workers & Manifest.json**
  - *Justificación:* Es la columna vertebral del proyecto. Permite que la aplicación se "instale" en la tablet y controle el tráfico de red, sirviendo activos desde la caché cuando no hay internet (cumpliendo el **RF-004**).
- **Almacenamiento Local: IndexedDB**
  - *Justificación:* A diferencia de localStorage (que es síncrono y limitado a 5MB), **IndexedDB** es asíncrono y permite almacenar grandes volúmenes de datos estructurados (catálogo completo de productos y ventas pendientes de sincronización) sin bloquear la interfaz de usuario.
- **Backend: Django 5.2.7**
  - *Justificación:* Django provee una estructura sólida y segura ("baterías incluidas").
- **Base de Datos: Azure SQL Server**
  - *Justificación:* Escalabilidad empresarial y compatibilidad nativa con el entorno de despliegue en Azure.

Patrones de Diseño Implementados

- **Patrón Offline-First:**
  - La aplicación no asume que hay internet. Primero intenta leer/escribir en la base de datos local (IndexedDB) y, en segundo plano, un proceso sincroniza con el servidor. Esto garantiza que la UI nunca se congele esperando una respuesta del servidor.
- **Arquitectura de Componentes (Atomic Design):**
  - Se estructura la UI en componentes reutilizables: *Átomos* (botones, inputs), *Moléculas* (tarjetas de producto), y *Organismos* (la grilla del POS). Esto mejora la mantenibilidad.
- **Patrón API Gateway / Service Layer (Frontend):**
  - En lugar de hacer llamadas fetch o axios dentro de los componentes, se centralizan en una carpeta de services/. Esto desacopla la vista de la lógica de red.

- *Ejemplo:* OrderService.saveOrder(order) maneja internamente si se guarda en local o se envía a la API.

2. **Modelo Vista Controlador (MVC) / MVT:**
   - En el backend (Django), se mantiene una clara separación entre los datos (Modelos), la lógica de presentación (Serializadores/Vistas) y las rutas (URLs).

Principales Decisiones Técnicas y su Impacto

Decisión	Impacto en la Solución
Uso de PWA en lugar de App Nativa (Android/iOS)	Reducción drástica de costos y tiempos. Permite tener un solo código fuente (Frontend) que funciona en cualquier dispositivo (Tablet Android, iPad, Laptop) sin pasar por las tiendas de aplicaciones.
Sincronización Diferida (Background Sync)	Resiliencia operativa. El Food Truck puede operar en zonas sin cobertura (eventos rurales) sin perder ventas. Los datos se suben solos cuando vuelve la señal.
Uso de JWT (JSON Web Token) para Auth	Seguridad Stateless. Al no depender de sesiones en el servidor, el token se puede almacenar localmente, permitiendo validar la "sesión" del cajero incluso si el servidor está inalcanzable temporalmente.
Infraestructura Serverless / IaaS (Azure App Service)	Menor carga de administración.

Tabla 7 – Decisiones Técnicas vs Impacto

Buenas Prácticas de Desarrollo Adoptadas

- **Seguridad (Security by Design):**
  - Implementación de **HTTPS** obligatorio para evitar intercepción de datos en redes públicas.
  - Uso de variables de entorno (.env) para no exponer credenciales de base de datos o claves secretas en el repositorio público.
  - Validación de datos tanto en Frontend (UX) como en Backend (Seguridad) para evitar inyecciones SQL o datos corruptos.
- **Mantenibilidad:**
  - **Código Limpio (Clean Code):** Uso de nombres de variables descriptivos (ej. calcularTotalVenta en vez de calc) y funciones pequeñas con una única responsabilidad.
  - **Estructura de Directorios Estándar:** Separación clara de carpetas: /components, /hooks, /context, /services, /assets.
- **Escalabilidad:**

- La arquitectura del sistema permite una escalabilidad transparente en Azure, de modo que si el negocio crece o aumenta la demanda, tanto la instancia del servidor como la base de datos pueden incrementar sus recursos sin necesidad de modificar la aplicación. Azure permite ajustar CPU, memoria, almacenamiento y rendimiento de la base de datos de forma vertical.
- El modelo de base de datos está normalizado (3FN), lo que asegura que el crecimiento de datos no genere redundancias ni anomalías.

16.4. Desarrollo y Soluciones Implementadas

A. Funcionalidades desarrolladas y su relación con los requerimientos

Funcionalidad Implementada	Requerimientos Cubiertos
Interfaz táctil optimizada para tablets que permite la selección rápida de productos, categorización visual y gestión del "Carrito de Compras" (agregar, quitar, modificar cantidad).	• RF-001 (Gestión de Pedidos) • RNF-003 (Usabilidad)
Implementación de Service Workers que interceptan las peticiones de red. Si no hay conexión, las transacciones se desvían y almacenan en IndexedDB localmente.	• RF-004 (Operación Offline) • RNF-004 (Disponibilidad)
Proceso en segundo plano (Background Sync) que detecta el restablecimiento de la red e inyecta las ventas almacenadas localmente hacia el servidor Azure mediante una cola FIFO (First-In, First-Out).	• RF-004-B (Sincronización) • RF-001-C (Integridad de Datos)
Modal dinámico que permite seleccionar variantes (ej. Tamaño) y extras (ej. Ingredientes adicionales) respetando reglas de exclusión lógica.	• RF-002 (Personalización) • RF-001-B (Modificación)
Funcionalidad para declarar el monto de apertura ("Fondo Fijo") y realizar el cierre (conteo de efectivo real vs. sistema), calculando diferencias automáticamente.	• RF-005 (Control de Caja) • RF-006 (Multiusuario)
Motor de renderizado que convierte el objeto "Venta" en un formato imprimible (Boleta/Comanda) compatible con impresoras térmicas de 80mm/58mm.	• RF-003 (Gestión Documentos) • RNF-002 (Cumplimiento SII)
Dashboard para la gestión de usuarios, roles, catálogo de productos y visualización de reportes de ventas consolidados.	• RF-006 (Roles) • RF-005-B (Reportes)

Tabla 8 - Funcionalidades vs Requerimientos

B. Retos enfrentados y soluciones aplicadas

Durante el desarrollo, enfrentamos desafíos técnicos relacionados al uso de una PWA y la implementación de la modalidad offline del sistema, además del uso de la infraestructura en la nube de Azure. A continuación, se describen los problemas y las soluciones aplicadas:

Reto 1: Curva de Aprendizaje y Configuración del PWA

- Problema: Al inicio del proyecto, el equipo no contaba con experiencia en el desarrollo sobre PWA.
- Solución Aplicada: Se abordó la brecha de conocimiento mediante un investigación y estudio técnico.

Reto 2: Configuración y Conexión de Servicios en Azure

- Problema: Debido a la poca experiencia en Azure, nos enfrentamos a dificultades al intentar replicar el entorno de desarrollo local en Azure. Al principio, no se lograba que la aplicación (Django) se comunicara con la base de datos o guardara las imágenes, ya que en la nube estos componentes no residen en la misma máquina, sino que son servicios independientes y separados.
- Solución Aplicada: Se optó por una implementación manual a través del Portal de Azure para entender y configurar los tres pilares fundamentales de la infraestructura:
    - Máquina Virtual (VM): Creación de un servidor básico con Linux para ejecutar el código de la aplicación.
    - Azure SQL Database: Creación de la instancia de base de datos relacional.
    - Azure Blob Storage: Creación de un espacio dedicado para almacenar las fotos de los productos.

Reto 3: Lograr gestionar los tiempos de cada integrante del equipo.

- Problema: No siempre se logró cumplir en tiempo y forma con las tareas dada las diversas contingencias personales de cada integrante del equipo.
- Solución Aplicada: Se debió trabajar en horarios extendidos para recuperar los atrasos y los temas pendientes.

C. Capturas o ejemplos de funcionamiento

Ilustración 13 - Login de Acceso

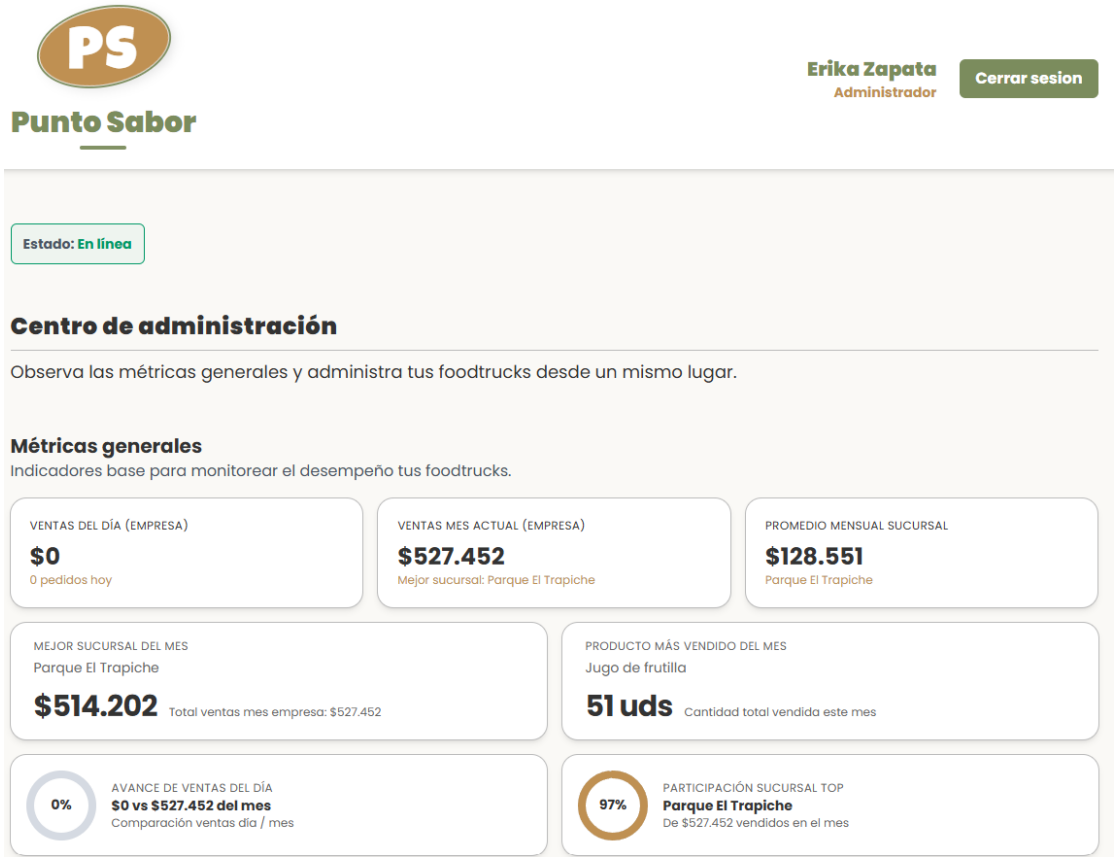

Ilustración 14 - Vista Administrador

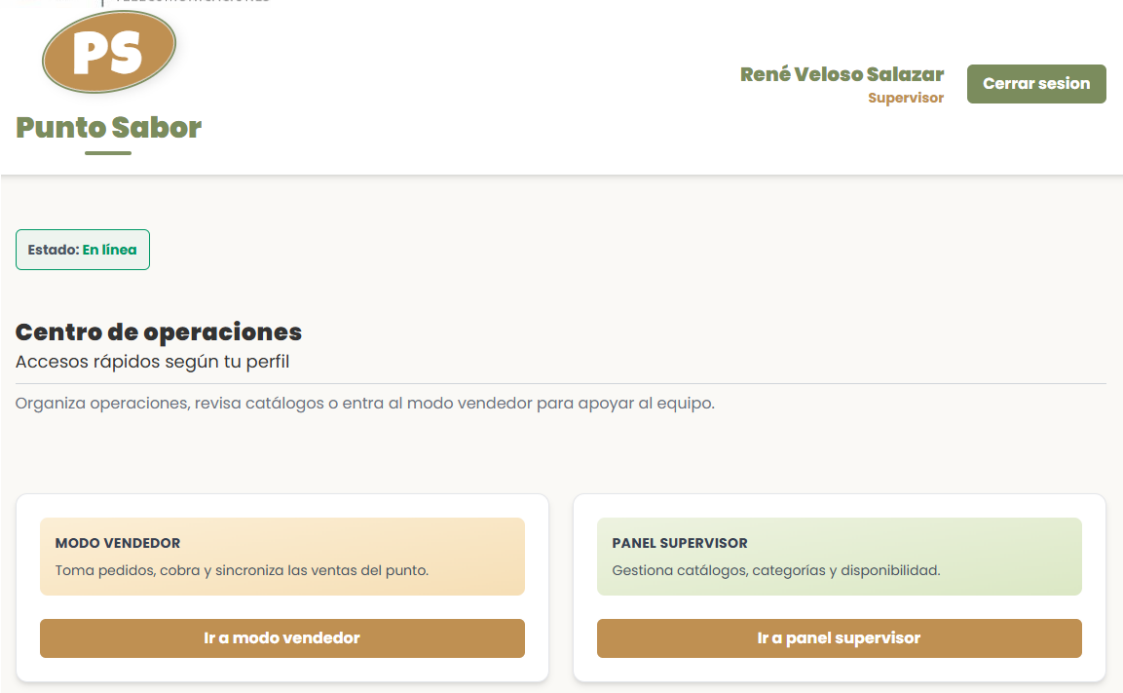

Ilustración 15 - Vista Supervisor

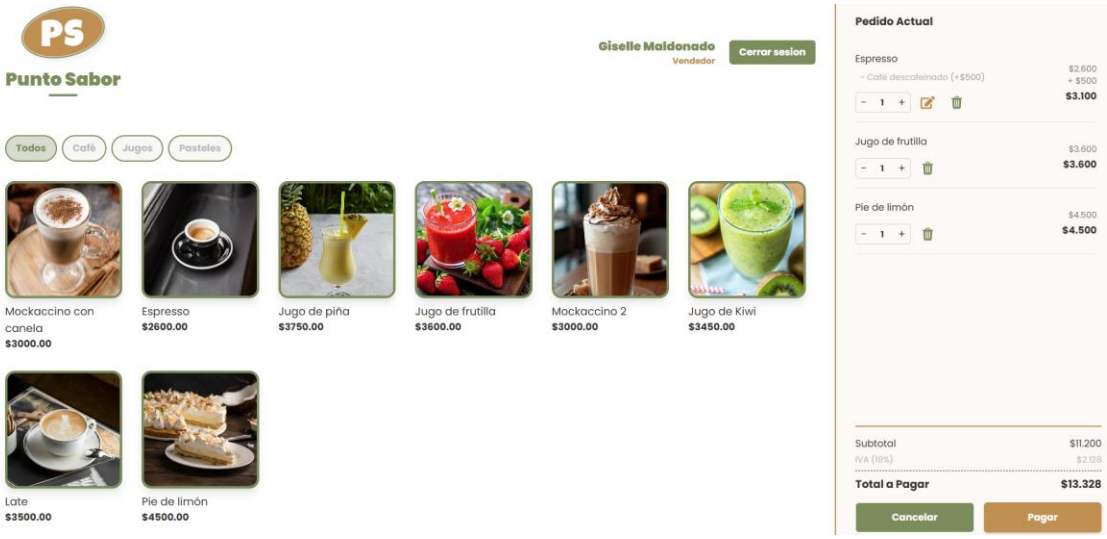

Ilustración 16 - Vista Vendedor

16.5. Pruebas y Validación del Sistema

A continuación, se detalla el proceso de aseguramiento de la calidad (QA) ejecutado al Sistema de Gestión para Food Truck. Los tipos de pruebas realizadas fueron las siguientes:

- Pruebas Unitarias: Para verificar la lógica interna de los componentes críticos del backend y frontend.
- Pruebas de Integración: Enfocadas en asegurar la correcta comunicación entre la PWA, la API y la base de datos, especialmente en los procesos de sincronización.
- Pruebas de Rendimiento: Para certificar los tiempos de respuesta y el cumplimiento de los estándares PWA (Offline/Online).
- Pruebas de Aceptación (UAT): Donde se validó el flujo de negocio.

Los resultados obtenidos se adjuntan en un documento Excel llamado “Planilla Casos de Prueba – Resultados.xlsx”

16.6. Despliegue y Entorno de Ejecución

A. Configuración del entorno

El sistema fue desarrollado bajo una arquitectura cliente-servidor de tres capas, utilizando tecnologías modernas y escalables:

- **Frontend**: React.js con PWA Toolkit, permitiendo una interfaz responsiva, táctil y funcional en modo offline mediante Service Workers e IndexedDB.
- **Backend**: Django (Python), encargado de la lógica de negocio, autenticación, sincronización y exposición de APIs RESTful.
- **Base de datos**: Azure SQL Server, con modelo relacional y soporte transaccional.
- **Almacenamiento local**: IndexedDB para persistencia offline y sincronización automática.
- **Impresión**: escpos-thermal-print (JavaScript) para integración con impresoras térmicas vía Bluetooth/USB.
- **Autenticación**: JWT (JSON Web Tokens) con cifrado ARGON2.
- **Gestión de tareas**: Azure DevOps, con tablero Kanban para seguimiento de sprints y backlog.

B. Diagrama de Infraestructura

La infraestructura se implementó en Microsoft Azure, utilizando los siguientes recursos:

- Máquina virtual (VM) para ejecución del backend.
- Red virtual (VNet), interfaz de red (NIC) y grupo de seguridad (NSG).
- IP pública para acceso externo.
- Base de datos SQL.
- Zona DNS para resolución de nombres.
- Punto de conexión privado para acceso seguro a recursos internos.

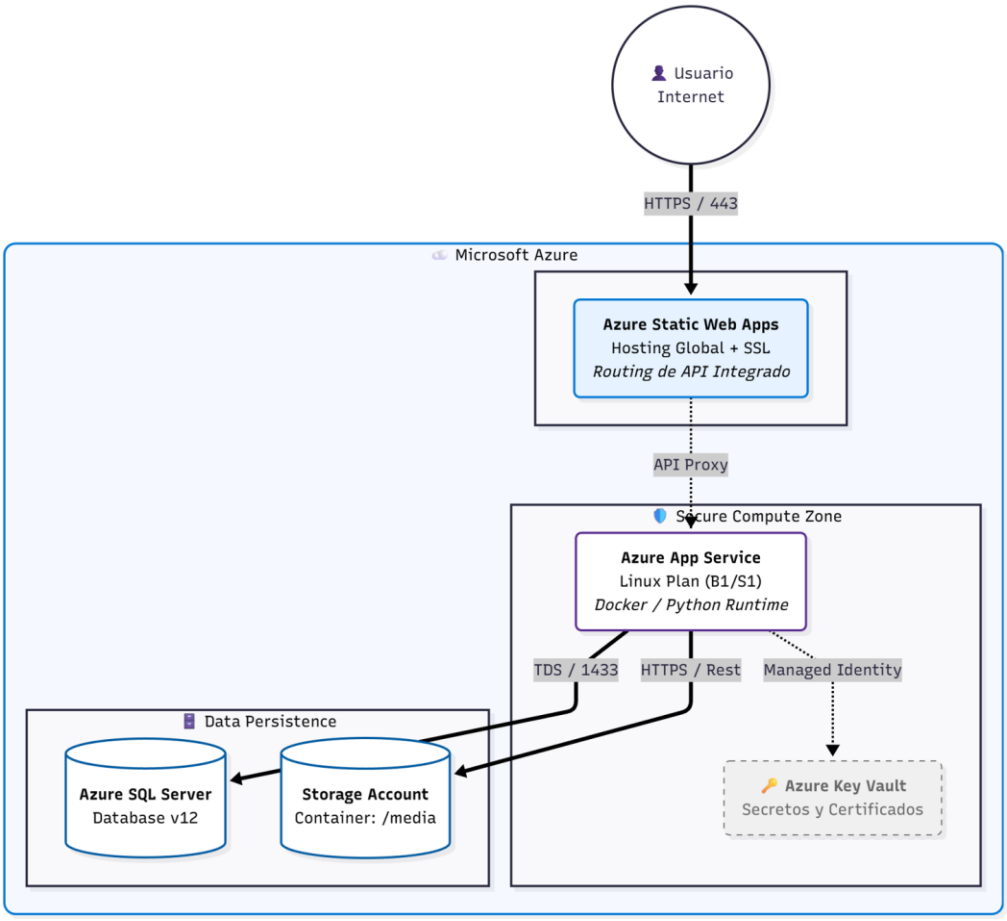

Ilustración 17 - Diagrama de Infraestructura

C. Instrucciones básicas para ejecutar o probar el sistema

Requisitos previos:

- Node.js y npm instalados para el frontend.
- Python 3.10+ y pip para el backend.
- Acceso a Azure SQL Server y credenciales JWT.
- Django para ejecutar el Backend

Pasos de ejecución:

1. Clonar el repositorio desde GitHub.
2. En el frontend:
    - cd frontend
    - bun install
    - bun run dev
3. En el backend:
    - cd backend
    - pip install -r requirements.txt
    - python manage.py runserver
4. Configurar variables de entorno (.env) con claves de API, credenciales y endpoints.
5. Acceder vía navegador a http://localhost:8000 para probar la interfaz.

16.7. Documentación Complementaria

A. Documentación de la API

La API RESTful expone endpoints para gestión de productos, pedidos, pagos y usuarios. Principios REST aplicados:

- GET: Lectura de recursos (/api/productos/)
- POST: Creación (/api/pedidos/)
- PUT/PATCH: Actualización (/api/pedidos/123/)
- DELETE: Eliminación (/api/productos/456/)

La autenticación se realiza mediante JWT en el encabezado Authorization: Bearer <token>. Para más detalles de la implementación se puede consultar la siguiente URL: https://foodtrucksapp.cl/api/docs/#/

B. Bitácora de implementación:

El siguiente es un resumen de los primeros versionamientos del código, para ver el detalle completo se puede acceder a la siguiente URL: **https://github.com/roydev-dw/capstone_grupo_2/commits/main**

Fecha	Sección	Descripción de la Implementación
01/10/2025	Backend	Subida inicial del backend del sistema
03/10/2025	Backend	Subida de modelos de base de datos
08/10/2025	Frontend	Mejora de componentes de vendedor y estilos
09/10/2025	Frontend	Implementación de carrito y carga de productos
10/10/2025	Frontend	Inicio de sesión con autenticación y manejo de errores
11/10/2025	Frontend	Cambio de color en PedidoActual
12/10/2025	Frontend	Ajustes menores
13/10/2025	Frontend	Lógica de carrito en Vendedor; mejora de estilos
14/10/2025	Frontend	Modal de opciones en carrito; ajuste de API
15/10/2025	Frontend	Cierre de pedido en PedidoActual; utilidades CSS
16/10/2025	Frontend	Mejora de estilos en PedidoActual y Login
18/10/2025	Frontend	Soporte para Dexie y manejo de carrito
19/10/2025	Frontend	Actualización de manifiesto PWA e iconos
20/10/2025	Frontend	Mejora en sincronización de productos y configuración de caché
21/10/2025	Frontend	Guías y plantillas APT; mejoras en carrito y configuración de Vite
21/10/2025	Backend	Creación de múltiples endpoints para lógica de negocio
25/10/2025	Frontend	Refactor del componente Vendedor con nueva estructura de API
26/10/2025	Frontend	Componente reutilizable Button y EstadoEnLinea
27/10/2025	Frontend	Actualización general

Tabla 9 - Bitácora de Implementación

16.8. Aspectos Éticos, Legales y de Seguridad

Consideraciones éticas

El sistema fue diseñado respetando principios de equidad, privacidad y transparencia. Se evita la recopilación excesiva de datos.

Cumplimiento de leyes y normas

Se cumple con la Ley 19.628 sobre protección de datos personales, asegurando:

- Minimización de datos recolectados.
- Acceso restringido a datos sensibles.

Mecanismos de autenticación y cifrado

- Autenticación mediante JWT con expiración y renovación segura.

- Cifrado ARGON2 para datos sensibles.
- Validación de roles y permisos por endpoint.

Uso responsable de inteligencia artificial

Aunque el sistema no incorpora IA directamente, se contempla su uso futuro para análisis de ventas y predicción de demanda, siempre bajo principios éticos y legales.

16.9. Conclusiones Técnicas y Aprendizajes

Principales logros técnicos

- Desarrollo de una PWA funcional con soporte offline.
- Integración con pasarela de pagos Transbank (Webpay).
- Despliegue completo en Azure con recursos interconectados.
- Validación positiva por usuarios piloto.

Dificultades encontradas y soluciones

- Sincronización offline: Resuelta con cola de operaciones y algoritmo Last-Write-Wins.
- Conectividad limitada: Mitigada con almacenamiento local y detección de estado de red.

Posibles mejoras o líneas futuras

- Integración con POS para pagos.
- Incorporación de IA para análisis predictivo.
- Panel administrativo con gestión avanzada.

17. Factibilidad Económica

17.1. Flujo de Caja

El objetivo del flujo de caja es proyectar los ingresos y egresos del proyecto para evaluar su liquidez y rentabilidad en el tiempo, lo que está totalmente asociado a la ejecución del proyecto dado que nuestra intención es comercializarlo en un futuro cercano, por lo cual es fundamental realizar las proyecciones necesarias que nos permitan evaluar la rentabilidad del mismo.

Los valores y datos utilizados para el flujo de caja son estimados, pero tratando de acercase a la realidad actual.

Año	Ingreso por Ventas	Costos	Utilidad Bruta	Impuestos	Utilidad Neta	Flujo de Caja
0	$0	$0	$0	$0	$0	**-$1,354,667** (inversión inicial + capital de trabajo)
1	$1,281,443	-$1,235,000	$46,442	-$11,610	$34,831	**$3,957**
2	$1,387,649	-$1,272,050	$115,598	-$28,900	$86,699	**$54,899**
3	$1,493,856	-$1,310,212	$183,644	-$45,911	$137,733	**$104,978**
…	…	…	…	…	…	…
10	$2,237,302	-$1,691,152	$546,150	-$136,538	$409,613	**$1,818,906**

Tabla 10 - Resumen Flujo de Caja

17.2. VAN

El objetivo del **Valor Actual Neto (VAN)** es saber si la suma de los flujos futuros, descontados a valor presente por una tasa elegida, al final deja ganancia después de recuperar la inversión. Es una medida clave para decidir si financieramente vale la pena invertir en el proyecto. En nuestro caso, evalúa si el sistema proyectado a 10 años paga la inversión inicial y deja una ganancia que justifique el riesgo y el costo de la inversión, lo que en nuestro caso es factible por resultar un **VAN de $235.202 que es mayor a cero.**

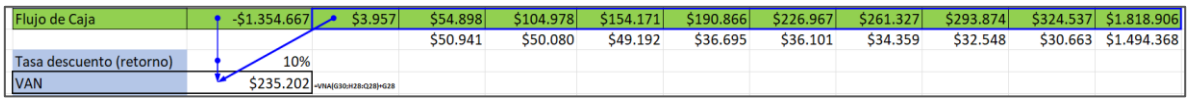

Ilustración 18 – Cálculo y Resultado VAN

17.3. TIR

El objetivo de la **Tasa Interna de Retorno (TIR)** es indicar la tasa de rentabilidad anualizada esperada del proyecto, comparando la rentabilidad del proyecto con una tasa de referencia, es decir, es la tasa de descuento que hace que el VAN sea igual a cero, si el TIR es mayor a la tasa de descuento, el proyecto es atractivo. En nuestro caso, tenemos un **TIR de 12%** lo que implica que el proyecto rinde un 12% anual sobre la inversión realizada, superando la tasa de oportunidad del 10%, lo cual lo hace financieramente viable.

Flujo de Caja	-$1.354.667	$3.957	$54.898	$104.978	$154.171	$190.866	$226.967	$261.327	$293.874	$324.537	$1.818.906
			$50.941	$50.080	$49.192	$36.695	$36.101	$34.359	$32.548	$30.663	$1.494.368
Tasa descuento (retorno)	10%										
VAN	$235.202										
TIR	12%	=TIR(G28:Q28)									

Ilustración 19 – Cálculo y Resultado TIR

18. Bibliografía y Referencias

Cámara Nacional de Comercio. (5 de Noviembre de 2025). *Ventas de comida rápida crecen un 3,9% anual durante el último trimestre, acumulando un alza del 3% entre enero y septiembre de este año*. Obtenido de https://cnc.cl/: https://cnc.cl/ventas-de-comida-rapida-crecen-un-39-anual-durante-el-ultimo-trimestre-acumulando-un-alza-del-3-entre-enero-y-septiembre-de-este-ano/

POSOFT. (2025). *Kit Punto de Venta*. Obtenido de Posoftcyf: https://www.posoftcyf.cl/kit-punto-de-venta-comida-rapida

19.Anexos

19.1. Fragmentos Código Frontend

```
import Dexie from 'dexie';

const DB_NAME = 'puntoSaborDB';
const LEGACY_DB_NAMES = ['appDB', 'foodTruckDB'];
const MIGRATION_FLAG = 'puntoSaborDB:migrated:v1';

export const db = new Dexie(DB_NAME);

db.version(1).stores({
  products:
    '&id, producto_id, categoria_id, updatedAt, pending, [categoria_id+updatedAt]',
  categories: '&id, categoria_id, updatedAt, pending',
  carrito: '&idItemCarrito',
  outbox: '++key, type, op, status, ts, tempId, targetId',
});

db.version(2)
  .stores({
    products:
      '&id, producto_id, categoria_id, updatedAt, pendingFlag, [categoria_id+updatedAt]',
    categories: '&id, categoria_id, updatedAt, pendingFlag',
    carrito: '&idItemCarrito',
    outbox: '++key, type, op, status, ts, tempId, targetId',
  })
  .upgrade(async (tx) => {
    await tx
      .table('products')
      .toCollection()
      .modify((item) => {
        item.pending = !!item.pending;
        item.pendingFlag = item.pending ? 1 : 0;
      });
    await tx
      .table('categories')
      .toCollection()
      .modify((item) => {
        item.pending = !!item.pending;
        item.pendingFlag = item.pending ? 1 : 0;
      });
  });
```

Ilustración 20 - Código Frontend 1

```css
@import 'tailwindcss';

@theme {
  --font-logo: 'Luckiest Guy', system-ui;
  --font-body: 'Poppins', sans-serif;

  --color-fondo: #faf9f6; /* Blanco hueso (fondo base) */
  --color-elemento: #ffffff; /* Blanco puro (tarjetas, contenedores) */
  --color-primario: #c1904d; /* Caramelo cálido */
  --color-secundario: #7b8c5b; /* Verde oliva complementario */
  --color-info: #4b6584; /* Azul grisáceo elegante (para Editar) */
  --color-peligro: #a45c5c; /* Rojo terracota (para Eliminar/Cancelar) */
  --color-texto: #333333; /* Gris oscuro para texto */
  --color-texto-suave: #666666; /* Gris medio para textos secundarios */
  --color-placeholder: #bcbcbc; /* Gris claro para inputs y detalles */
  --color-drag: #74d4ff; /* Gris muy claro para áreas de arrastre */
}

body {
  font-family: var(--font-body);
  color: var(--color-texto);
  background-color: var(--color-fondo);
}

@layer utilities {
  .grid-cols-extra {
    @media (min-width: 2000px) {
      grid-template-columns: repeat(8, minmax(0, 1fr));
    }
  }
}
```

Ilustración 21 - Código Frontend 2

```javascript
function toWebpFile(file, { maxWidth = 1600, maxHeight = 1600, quality = 82 } = {}) {
  return new Promise((resolve, reject) => {
    try {
      Resizer.imageFileResizer(
        file,
        maxWidth,
        maxHeight,
        'WEBP',
        quality,
        0,
        (uri) => {
          const base = (file.name || 'upload').replace(/\.[^.]+$/, '');
          resolve(fileFromDataUrl(uri, `${base}.webp`, 'image/webp'));
        },
        'base64'
      );
    } catch (err) {
      reject(err);
    }
  });
}
```

Ilustración 22 - Código Frontend 3

```
{
  "name": "frontend",
  "private": true,
  "version": "0.0.0",
  "type": "module",
  "scripts": {
    "dev": "vite",
    "build": "vite build",
    "lint": "eslint .",
    "preview": "vite preview",
    "docs": "jsdoc -c jsdoc.json"
  },
  "dependencies": {
    "@fontsource/luckiest-guy": "^5.2.8",
    "@fontsource/poppins": "^5.2.6",
    "@hookform/resolvers": "^5.2.2",
    "@tailwindcss/vite": "^4.1.13",
    "dexie": "^4.2.1",
    "dexie-react-hooks": "^4.2.0",
    "jsdoc": "^4.0.5",
    "prop-types": "^15.8.1",
    "react": "18",
    "react-dom": "18",
    "react-hook-form": "^7.66.0",
    "react-hot-toast": "^2.6.0",
    "react-icons": "^5.5.0",
    "react-image-file-resizer": "^0.4.8",
    "react-router-dom": "^6",
    "tailwindcss": "^4.1.13",
    "yup": "^1.7.1"
  },
  "devDependencies": {
    "@eslint/js": "^9.33.0",
    "@types/react": "18",
    "@types/react-dom": "18",
    "@vitejs/plugin-react": "^5.0.0",
    "autoprefixer": "^10.4.21",
    "eslint": "^9.33.0",
    "eslint-plugin-react-hooks": "^5.2.0",
    "eslint-plugin-react-refresh": "^0.4.20",
    "globals": "^16.3.0",
    "postcss": "^8.5.6",
    "vite": "^7.1.2",
    "vite-plugin-pwa": "^1.1.0"
  }
}
```

Ilustración 23 - Código Frontend 4

```
import { defineConfig } from 'vite';
import react from '@vitejs/plugin-react';
import tailwindcss from '@tailwindcss/vite';
import { VitePWA } from 'vite-plugin-pwa';

export default defineConfig({
  plugins: [
    react(),
    tailwindcss(),
    VitePWA({
      registerType: 'autoUpdate',
      devOptions: { enabled: false },

      includeAssets: [
        'favicon.svg',
        'favicon.ico',
        'robots.txt',
        'apple-touch-icon.png',
      ],
      manifest: {
        name: 'Punto Sabor FoodTruck',
        short_name: 'Punto Sabor',
        description: 'App de ventas para FoodTruck',
        icons: [
          { src: 'icon-48x48.png', sizes: '48x48', type: 'image/png' },
          { src: 'icon-72x72.png', sizes: '72x72', type: 'image/png' },
          { src: 'icon-96x96.png', sizes: '96x96', type: 'image/png' },
          { src: 'icon-128x128.png', sizes: '128x128', type: 'image/png' },
          { src: 'icon-144x144.png', sizes: '144x144', type: 'image/png' },
          { src: 'icon-152x152.png', sizes: '152x152', type: 'image/png' },
          { src: 'icon-192x192.png', sizes: '192x192', type: 'image/png' },
          { src: 'icon-256x256.png', sizes: '256x256', type: 'image/png' },
          { src: 'icon-384x384.png', sizes: '384x384', type: 'image/png' },
          { src: 'icon-512x512.png', sizes: '512x512', type: 'image/png' },
        ],
        start_url: '/',
        display: 'standalone',
        background_color: '#ffffff',
        theme_color: '#000000',
      },

      workbox: {
        cleanupOutdatedCaches: true,
        globPatterns: ['**/*.{js,css,html,ico,png,svg,json,woff,woff2}'],
        navigateFallback: 'index.html',

        runtimeCaching: [
          {
            handler: 'CacheFirst',
            urlPattern:
              /^https:\/\/devsapihub\.com\/img-fast-food\/.*\.(?:png|jpg|jpeg|webp|avif|svg)$/i,
            options: {
              cacheName: 'api-imagenes',
              expiration: {
                maxEntries: 60,
                maxAgeSeconds: 60 * 60 * 24 * 7,
              },
              cacheableResponse: { statuses: [0, 200] },
              rangeRequests: true,
            },
          },
          {
            handler: 'StaleWhileRevalidate',
            urlPattern: /^https:\/\/devsapihub\.com\/api-fast-food(?:\/|$)/i,
            options: {
              cacheName: 'api-json',
              expiration: {
                maxEntries: 30,
                maxAgeSeconds: 60 * 60,
              },
              cacheableResponse: { statuses: [200] },
            },
          },
        ],
      },
    }),
  ],
});

```

Ilustración 24 - Código Frontend 5

19.2. Fragmentos Código Backend

```python
from rest_framework.authentication import BaseAuthentication
from rest_framework.exceptions import AuthenticationFailed
from core.jwt_utils import decode_token
from core.models import Usuario

class CustomJWTAuthentication(BaseAuthentication):
    """
    Autenticación JWT para DRF + Swagger.
    Lee el encabezado Authorization: Bearer <token>.
    """

    keyword = "Bearer"

    def authenticate(self, request):
        auth_header = request.headers.get("Authorization")

        if not auth_header:
            return None  # DRF sigue buscando otros métodos

        if not auth_header.startswith("Bearer "):
            raise AuthenticationFailed("Formato inválido. Use 'Bearer <token>'")

        token = auth_header.split(" ")[1]

        try:
            payload = decode_token(token)
        except Exception:
            raise AuthenticationFailed("Token inválido o expirado")

        # Buscar usuario
        try:
            user = Usuario.objects.get(pk=payload["sub"])
        except Usuario.DoesNotExist:
            raise AuthenticationFailed("Usuario no encontrado")

        # Guardar payload para usar en request.jwt_payload
        request.jwt_payload = payload

        return (user, None)
```

Ilustración 25 - Código Backend 1

```python
from argon2 import PasswordHasher

ph = PasswordHasher()

def make_hash(raw_password: str) -> str:
    return ph.hash(raw_password)

def check_password(raw_password: str, hashed_password: str) -> bool:
    try:
        return ph.verify(hashed_password, raw_password)
    except Exception:
        return False
```

Ilustración 26 - Código Backend 2

```python
from django.db import models

class Empresa(models.Model):
    empresa_id = models.AutoField(primary_key=True, db_column='EmpresaId')
    nombre = models.CharField(max_length=100, db_column='Nombre')
    rut = models.CharField(max_length=12, unique=True, db_column='RUT')
    direccion = models.CharField(max_length=200, db_column='Direccion', blank=True, null=True)
    telefono = models.CharField(max_length=15, db_column='Telefono' , blank=True, null=True)
    email = models.EmailField(max_length=100, db_column='Email')
    fecha_creacion = models.DateTimeField(auto_now_add=True, db_column='FechaCreacion')
    estado = models.BooleanField(default=True, db_column='Estado')

class Sucursal(models.Model):
    sucursal_id = models.AutoField(primary_key=True, db_column='SucursalId')
    empresa = models.ForeignKey('Empresa', on_delete=models.CASCADE, db_column='EmpresaId')
    nombre = models.CharField(max_length=100, db_column='Nombre')
    direccion = models.CharField(max_length=200, db_column='Direccion' , blank=True, null=True)
    telefono = models.CharField(max_length=15, db_column='Telefono', blank=True, null=True)
    estado = models.BooleanField(default=True, db_column='Estado')
    fecha_creacion = models.DateTimeField(auto_now_add=True, db_column='FechaCreacion')

    class Meta:
        db_table = 'Sucursales'
        verbose_name = 'Sucursal'
        verbose_name_plural = 'Sucursales'
        ordering = ['-fecha_creacion']

    def __str__(self):
        return f'{self.nombre} - ({self.empresa.nombre})'


class Rol(models.Model):
    rol_id = models.AutoField(primary_key=True, db_column='RolId')
    empresa = models.ForeignKey('Empresa', on_delete=models.CASCADE, db_column='EmpresaId', null=True, blank=True)
    nombre = models.CharField(max_length=50, db_column='Nombre')
    descripcion = models.TextField(db_column='Descripcion', blank=True, null=True)
    fecha_creacion = models.DateTimeField(auto_now_add=True, db_column='FechaCreacion')
```

Ilustración 27 - Código Backend 3

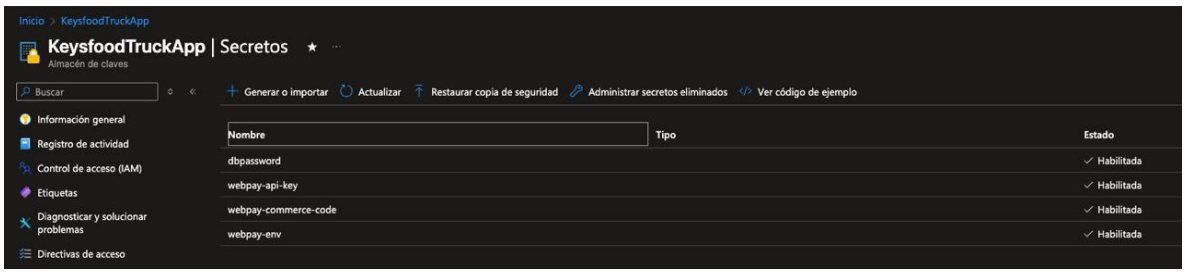

Ilustración 28 - Código Backend 4