Gómez Pizarro, Gonzalo

Informe de Proyecto Integrador - Cantina UCC

Tesis para la obtención del título de grado de Ingeniería de Sistemas

Directores:

Porrini, Federico Eduardo

Carreño, Ignacio Luciano

Di Marco, Octavio

Universidad Católica de Córdoba
Facultad de Ingeniería

Proyecto: Cantina UCC

Informe Final de Grado

Alumnos:

- Gómez Pizarro, Gonzalo

Directores:

- Porrini, Federico Eduardo
- Carreño, Ignacio Luciano
- Di Marco, Octavio

xxx de <mes> de 2025

Córdoba - Argentina

ÍNDICE

RESUMEN - ABSTRACT

Español

Este proyecto de grado aborda la problemática de las largas filas y los tiempos de espera en la cantina de la Universidad Católica de Córdoba (UCC), que afectan negativamente la experiencia de estudiantes y docentes. Para resolverlo, se desarrolló una aplicación web responsive con el objetivo de optimizar integralmente el proceso de compra y gestión de pedidos. A lo largo de este informe, se detalla cada fase del proyecto, desde el diagnóstico inicial y la definición de objetivos hasta la implementación técnica y el análisis de sus impactos.

El documento comienza presentando el marco teórico que sustenta la solución, incluyendo un análisis de campo con la administración de la cantina y un sondeo de la percepción de los usuarios, así como un estudio de soluciones similares en el mercado y las tecnologías evaluadas. Posteriormente, se expone la propuesta de solución, describiendo el alcance funcional a través de historias de usuario, el diseño de la interfaz y la arquitectura de microservicios elegida. Esta arquitectura, desplegada en Google Cloud Platform (GCP), utiliza servicios clave como Firestore, Cloud Run y Cloud Storage, con un frontend desarrollado en Next.js y un backend en Python con FastAPI.

Como resultado, se obtuvo una plataforma completamente funcional que permite a los usuarios realizar compras anticipadas y gestionar planes de comida, mientras que los administradores pueden gestionar productos, órdenes y resúmenes de ventas. Finalmente, el informe analiza en profundidad el impacto multidimensional del proyecto:

- Impacto Económico: Se presenta un análisis detallado de los costos operativos bajo diferentes escenarios de uso, demostrando la viabilidad y escalabilidad del sistema.
- Impacto Social y RSU: Se explora cómo la solución se alinea con los principios de Responsabilidad Social Universitaria, mejorando la calidad de vida en el campus.
- Impacto Medioambiental: Se cuantifica el beneficio ecológico derivado de la eliminación de tickets impresos y la reducción del desperdicio de alimentos.

En su conjunto, este documento no solo presenta la resolución de una necesidad operativa, sino que también sirve como una guía completa del ciclo de vida del desarrollo de una aplicación web moderna, evaluando sus efectos desde una perspectiva integral.

English

This final degree project addresses the problem of long queues and waiting times at the Catholic University of Córdoba (UCC) canteen, which negatively affects the experience of students and faculty. To solve this, a responsive web application was developed to comprehensively optimize the purchasing and order management process. Throughout this report, each phase of the project will be detailed, from the initial diagnosis and objective definition to the technical implementation and analysis of its impacts.

The document begins by presenting the theoretical framework that supports the solution,

including a field analysis with the canteen's administration and a survey of user perceptions, as well as a study of similar market solutions and the technologies evaluated. Subsequently, the proposed solution is presented, describing the functional scope through user stories, the interface design, and the chosen microservices architecture. This architecture, deployed on Google Cloud Platform (GCP), utilizes key services such as Firestore, Cloud Run, and Cloud Storage, with a frontend developed in Next.js and a backend in Python with FastAPI.

The result is a fully functional platform that allows users to make advance purchases and manage meal plans, while administrators can manage products, orders, and sales summaries. Finally, the report thoroughly analyzes the multidimensional impact of the project:

- Economic Impact: A detailed analysis of operating costs under different usage scenarios is presented, demonstrating the system's viability and scalability.
- Social Impact and USR: It explores how the solution aligns with the principles of University Social Responsibility, improving the quality of life on campus.
- Environmental Impact: The ecological benefit derived from eliminating printed tickets and reducing food waste is quantified.

As a whole, this document not only presents the resolution of an operational need but also serves as a comprehensive guide to the lifecycle of modern web application development, evaluating its effects from an integral perspective.

PRESENTACIÓN DEL TEMA

El proyecto **Cantina UCC** nace de una necesidad tangible y cotidiana observada en el corazón de la vida universitaria: las extensas filas y los prolongados tiempos de espera en la cantina de la Universidad Católica de Córdoba (UCC), particularmente durante las horas pico del almuerzo. Esta problemática fue identificada de primera mano por un miembro del equipo, quien experimentó cómo esta ineficiencia afectaba negativamente la experiencia de estudiantes y docentes, consumiendo una parte valiosa de su tiempo de descanso y estudio.

En su concepción inicial, el proyecto buscaba ser una solución directa y enfocada: agilizar el proceso de compra de los menús diarios para reducir las demoras. Sin embargo, un análisis más profundo durante la fase de planificación reveló una oportunidad mucho mayor. Desarrollar un sistema para optimizar las compras no sólo resolvería el problema de las filas, sino que, sin añadir una complejidad excesiva, podría sentar las bases para una modernización integral del servicio.

De esta manera, Cantina UCC evolucionó de ser una simple herramienta de gestión de pedidos a una propuesta de valor completa. El objetivo se expandió para transformar la operatividad de la cantina, ofreciendo una plataforma digital que no solo optimiza los tiempos, sino que también mejora la planificación de la demanda, introduce nuevas modalidades como los "Planes de Comida" y digitaliza el proceso de pago. El proyecto se convierte así en una solución integral que busca modernizar el servicio, brindando una experiencia más eficiente, cómoda y accesible tanto para la comunidad universitaria como para los administradores del servicio.

GLOSARIO

Proveedores de la cantina: La Universidad Católica de Córdoba terceriza el servicio de la cantina, por lo que los proveedores pueden cambiar con el tiempo y deberán adaptarse al uso del sistema. La idea es que la adaptación sea sencilla y comprensible, para que el sistema siga siendo utilizado de manera efectiva a lo largo de los años.

Planes de Comida: Sistema que permite a los usuarios la compra anticipada de menús con descuento, fomentando la fidelización de clientes. Los administradores pueden crear y gestionar estos planes , y los usuarios pueden adquirirlos y utilizarlos a través de la aplicación.

Pedido Anticipado: Funcionalidad principal de la aplicación que permite a los usuarios realizar y pagar sus pedidos con antelación. Esto tiene como objetivo reducir las filas y los tiempos de espera en la cantina, mejorando la experiencia del usuario.

Usuario Invitado: Término para un usuario que navega por la aplicación sin registrarse. Este tipo de usuario puede explorar el catálogo de productos y agregarlos al carrito de compras.

Usuario Registrado: Usuario que ha creado una cuenta en la aplicación, ya sea mediante correo electrónico o una cuenta de Google. Este nivel de acceso permite realizar compras, gestionar planes de comida y ver el historial de pedidos.

PreOrden: Registro temporal de un pedido que se crea antes de que el pago sea confirmado. Una vez que el pago se realiza con éxito, la información de la "PreOrden" se transfiere a una orden definitiva y la "PreOrden" se elimina. Las "PreÓrdenes" no pagadas se eliminan diariamente.

DIAGNÓSTICO (PROBLEMÁTICA)

Estado del Arte

En la Universidad Católica de Córdoba (UCC), la cantina cumple un rol fundamental al proveer alimentos y bebidas a estudiantes, docentes y personal administrativo. Sin embargo, durante los horarios de mayor afluencia, especialmente al mediodía, el servicio se ve afectado por largas filas y tiempos de espera prolongados. Este problema se debe a la acumulación de pedidos simultáneos y al proceso de pago manual, lo que ralentiza la atención y genera congestión en el área de la cantina.

Actualmente, los usuarios deben realizar su compra de manera presencial, seleccionando su menú, esperando en fila para realizar el pago y luego retirando su pedido. Esta dinámica genera demoras que afectan la experiencia del usuario, ya que gran parte de su tiempo de descanso se pierde en la espera. En particular, los estudiantes que cuentan con recreos cortos (por ejemplo, de 20 minutos) muchas veces no tienen tiempo suficiente para hacer la fila, lo que los obliga a optar por no comprar en la cantina o a llegar tarde a sus clases. Sin embargo, si pudieran encargar y pagar su pedido con anticipación, tendrían la posibilidad de retirarlo rápidamente al comienzo de su recreo y disponer del tiempo suficiente para comer con tranquilidad.

Si bien existen soluciones digitales en el mercado para la gestión de pedidos y pagos en comercios gastronómicos, la cantina de la UCC aún no cuenta con una plataforma que facilite este proceso. La implementación de un sistema digital podría representar una oportunidad significativa para optimizar la operación, modernizar el servicio y mejorar la satisfacción de los clientes.

Impacto

La situación actual genera diversas consecuencias que afectan tanto a los clientes como a la administración de la cantina:

<u>Para los estudiantes y docentes:</u>

- Pérdida de tiempo en filas extensas, reduciendo el tiempo efectivo de descanso o estudio.
- Frustración y desmotivación al tener que esperar largos períodos sólo para comprar un almuerzo.
- Imposibilidad de comprar comida en la cantina para aquellos con recreos cortos, ya que no tienen tiempo suficiente para hacer la fila.
- Limitaciones en la disponibilidad de ciertos productos debido a la alta demanda concentrada en poco tiempo.

<u>Para la administración de la cantina:</u>

- Dificultad en la gestión eficiente de los pedidos en momentos de alta demanda.

- Posible disminución en las ventas debido a clientes que deciden no comprar por la espera prolongada.

- Falta de herramientas para monitorear y prever la demanda de ciertos productos.

Ante esta problemática, surge la necesidad de implementar una solución tecnológica que permita agilizar el proceso de compra y pago, mejorando la experiencia de los usuarios y optimizando la operatividad de la cantina. Un sistema de pedidos anticipados permitiría que los estudiantes con poco tiempo disponible puedan simplemente retirar su pedido sin necesidad de hacer filas, haciendo que la experiencia de compra sea mucho más eficiente y accesible.

OBJETIVOS

Objetivo Global

Desarrollar una web app responsive que optimice la gestión y compra en la cantina de la Universidad Católica de Córdoba, permitiendo a los usuarios realizar pedidos y pagos de manera anticipada. De esta forma, se busca reducir los tiempos de espera, mejorar la eficiencia operativa del servicio y brindar una experiencia más ágil y accesible tanto para estudiantes y docentes como para la administración de la cantina.

Objetivos Específicos

✅ Desarrollar una interfaz para que el cliente pueda explorar, elegir y comprar los productos ofrecidos por la cantina, comprar planes de comida y hacer uso de los mismos y ver el estado de sus órdenes

✅ Diseñar e implementar una interfaz intuitiva y fácil de usar, asegurando una experiencia óptima para clientes y administradores.

✅ Desarrollar un sistema de administración que permita la gestión de pedidos, productos, planes de comidas y resúmenes de compras diarias.

✅ Integrar una pasarela de pago, para que los usuarios puedan realizar compras de manera rápida y segura sin necesidad de pagar en efectivo.

MARCO TEÓRICO

El presente Marco Teórico aborda la problemática de las largas filas en la cantina de la Universidad Católica de Córdoba (UCC) durante los horarios de almuerzo. Esta situación afecta negativamente la experiencia de estudiantes y docentes, generando demoras y disminuyendo la eficiencia del servicio. A continuación, se exploran diversos aspectos relacionados con esta problemática, incluyendo el contexto general, análisis de campo, opciones similares en el mercado y un análisis de algunas posibles tecnologías sobre las que se implementaría el sistema.

1. Contexto General del Problema

La eficiencia en los sistemas de prestación de servicios es un campo de estudio fundamental en la gestión de operaciones. Un fenómeno central en este ámbito es la gestión de colas de espera, cuya optimización impacta directamente en la percepción de calidad del servicio y la satisfacción del cliente. En entornos con picos de demanda concentrados en breves períodos, como los servicios de alimentación universitarios, la formación de largas filas no es solo una molestia, sino un cuello de botella operativo que degrada la experiencia del usuario y puede generar pérdidas económicas.

Desde una perspectiva teórica, este problema se analiza a través de la Teoría de Colas, que modela los conflictos entre la demanda de un servicio y la capacidad para proveerlo. Una gestión ineficiente, basada en procesos manuales y secuenciales (seleccionar, pagar, retirar), maximiza los tiempos de espera y reduce el rendimiento del sistema. Esta ineficiencia es particularmente crítica en contextos donde el tiempo del usuario es un recurso escaso y no renovable, como lo es el receso entre clases para un estudiante.

La solución a esta problemática se encuentra en la transformación digital de los procesos de servicio. La implementación de sistemas de pedidos anticipados representa un cambio de paradigma: se transita de un modelo reactivo, donde el servicio comienza cuando el cliente llega, a un modelo proactivo, donde la orden se gestiona antes del pico de demanda. Esta estrategia no solo optimiza el flujo de trabajo interno, sino que fundamentalmente desacopla el proceso de pago y decisión del proceso de retiro, eliminando las principales causas de congestión y redefiniendo la experiencia del cliente hacia un modelo más ágil y eficiente.

2. Análisis de Campo

Se realizó un análisis de campo breve centrado en los dos actores principales del ecosistema de la cantina: la administración y los usuarios finales (estudiantes y docentes). Este enfoque dual permitió obtener una comprensión integral de las necesidades y expectativas desde ambas perspectivas.

2.1. Perspectiva de la Administración: La Entrevista Clave

Se llevó a cabo una entrevista directa con el administrador de la cantina de la UCC, una acción fundamental que proporcionó una visión interna y detallada de la operación actual. Durante la conversación, el administrador validó el diagnóstico del equipo, reconociendo

que la congestión se debe al proceso manual y simultáneo de pedido y pago, que crea un "cuello de botella" y ralentiza el servicio.

El administrador mostró un claro interés en la adopción de un sistema de pedidos anticipados , pero estableció un requisito crítico y no negociable para su viabilidad: la garantía total de la recepción efectiva del pago antes de que el personal comience a procesar cualquier pedido. Esta condición, más que una simple preferencia, se presentó como un pilar fundamental para la implementación del sistema.

Ante esta exigencia, el equipo explicó que existen mecanismos técnicos que permiten validar automáticamente los pagos antes de liberar los pedidos. Por ejemplo, mediante la comunicación (vía *webhook*) entre la plataforma de cobro y el sistema de gestión. La referencia a estos mecanismos no implicó una decisión tecnológica en esta etapa, sino que respondió a la necesidad de ilustrar al cliente sobre la viabilidad de su requerimiento.

Adicionalmente, el administrador especificó otros requisitos operativos esenciales para adaptar la solución a su flujo de trabajo:

- La necesidad de que el sistema genere automáticamente informes diarios de compras y montos para facilitar el registro de movimientos y el cierre de caja.
- La funcionalidad de poder imprimir los tickets (comandas) de los pedidos confirmados para ser procesados por la cocina.

Finalmente, se sondeó la idea de implementar "Planes de Comida" para la compra anticipada de múltiples menús con descuento. El administrador acogió esta propuesta positivamente, indicando que se alineaba con las prácticas de descuento actuales del negocio y que digitalizar su gestión representaría una mejora significativa.

Ver Anexo 1

2.2. Percepción de los Usuarios: Encuesta de Validación

Para validar cuantitativamente la problemática diagnosticada y medir la aceptación de una potencial solución digital, se realizó una encuesta formal dirigida a la comunidad universitaria, la cual obtuvo un total de 224 respuestas. Los resultados no solo confirman la necesidad de una intervención, sino que dimensionan la magnitud del problema.

El problema de las filas es una experiencia generalizada y medible. Un **71,7%** de los encuestados afirmó haber hecho fila por más de 10 minutos en la cantina para cualquier tipo de compra (por lo menos alguna vez). Al analizar específicamente el proceso del almuerzo (entre quienes consumen), los tiempos de espera percibidos son significativos: un 37,1% reporta una espera de 10 minutos, un 13,2% de 15 minutos y un 1.8% de más de 20 minutos. Esto indica que para el 52,1% de los consumidores de almuerzos, el proceso completo demanda 10 minutos o más. Al ponderar los tiempos de espera según su probabilidad reportada en la encuesta, se obtiene una demora promedio de **8,2 minutos por comida**.

Estas demoras tienen un impacto directo en la decisión de compra. Entre los que consumen almuerzos regularmente un **82.1%** de los encuestados admitió haber decidido **NO comprar un almuerzo** en alguna ocasión debido a la fila que había. Esta cifra de abandono es aún más drástica en compras generales (ej. cafés, snacks), donde un **88.8%** de los 224

encuestados ha optado por no comprar algo por no querer o no poder esperar.

La receptividad hacia una potencial solución tecnológica es abrumadoramente positiva. Ante la pregunta de si usarían un sistema para ver el menú, pagar con anticipación y retirar en un horario definido, un **96.4%** de los 224 encuestados respondió afirmativamente.

Las 57 respuestas abiertas de la encuesta proporcionaron un contexto cualitativo valioso, reforzando el entusiasmo por la idea (con frases como "Hagan realidad el sueño de muchos", "me parece una fantástica idea" y "Sería increíble"). De manera crucial, varias respuestas identificaron con precisión el punto de dolor central: el cuello de botella no está en la preparación de la comida, sino en el proceso de pago. Los usuarios señalaron explícitamente que "la fila es por la demora de la caja" y que "el sistema de cobranza de la cantina es muuuuuy lento". Adicionalmente, muchas de las funcionalidades deseadas por los usuarios se alinean directamente con los objetivos de una solución de pedidos anticipados. Surgieron solicitudes espontáneas para "poder consultar los menús disponibles desde cualquier lado", "un plan mensual de pago" con beneficios, e incluso la idea de "cargar el día anterior que va a comer" para que la cocina pueda planificar la producción. Finalmente, las respuestas también reflejaron una insatisfacción general con aspectos del servicio actual (precios, calidad, variedad, trato), lo que sugiere una alta disposición de los usuarios a adoptar un nuevo sistema que mejore su experiencia general.

Por lo tanto, los datos de la encuesta proveen una validación robusta. Demuestran que el problema de las filas es real y frecuente, que genera una pérdida de ventas tangible para la cantina y que la comunidad universitaria está dispuesta a adoptar masivamente (96.4%) una solución digital que resuelva estos problemas.

Podrán encontrar la encuesta en el **anexo 2**

2.3. Conclusión del Análisis

Las acciones llevadas a cabo en el análisis de campo confirman con solidez tanto la viabilidad como la necesidad de una solución tecnológica. La viabilidad operativa queda establecida por la disposición de la administración a adoptar el sistema bajo condiciones claras. Por otro lado, la necesidad del usuario ya no es una hipótesis de "demanda latente", sino un hecho validado cuantitativamente: los datos de 223 encuestados demuestran que el problema de las filas es una experiencia real (impactando al 88.8% en compras generales) y que existe una abrumadora disposición (96.4%) para adoptar una solución de pedidos anticipados.

En conjunto, estos hallazgos justifican plenamente el desarrollo de la plataforma, asegurando que responde a una necesidad sentida y que será bien recibida por toda la comunidad universitaria.

3. Opciones Similares en el Mercado

El mercado actual ofrece una variedad de soluciones digitales para la gestión de pedidos y pagos en entornos gastronómicos, cada una con sus propias características, ventajas y desventajas. Analizar estas opciones permite comprender el panorama competitivo y el nicho específico que el proyecto **Cantina UCC** busca llenar.

Entre las soluciones existentes, se pueden identificar principalmente tres categorías:

- **Aplicaciones de Delivery de Terceros:** Plataformas como Uber Eats, Rappi, DiDi Food, Glovo o Just Eat son ampliamente conocidas y ofrecen un servicio de logística de entrega externa. Su principal atractivo radica en su vasta base de clientes, lo que permite a los restaurantes aumentar significativamente sus ventas y su alcance sin una inversión directa en marketing. Para el cliente, estas aplicaciones brindan una gran comodidad y conveniencia, permitiendo pedir comida desde cualquier lugar y recibirla rápidamente. Además, la plataforma externa se encarga de la logística de entrega, los pagos y la gestión de riesgos asociados. Sin embargo, estas ventajas vienen acompañadas de desventajas significativas, como las altas comisiones que cobran (que pueden ascender hasta el 30% de las ventas), lo que impacta directamente en los márgenes de ganancia del negocio. Los restaurantes también pueden desarrollar una dependencia y un control limitado sobre ciertos aspectos de su operación y la calidad del servicio, ya que los retrasos o errores pueden afectar su reputación, incluso si están fuera de su control.

- **Software de Gestión de Restaurantes:** Sistemas integrales que van más allá de la simple toma de pedidos. Estas plataformas suelen incluir funcionalidades avanzadas como gestión de inventarios (control de recetas, ingredientes), facturación, sistemas de punto de venta (POS), gestión de nóminas, seguimiento del rendimiento, programas de fidelización y análisis de ventas detallados. Algunos se especializan en agilizar operaciones, mejorar la experiencia del cliente, optimizar el procesamiento de pagos o gestionar pedidos omnicanal. La ventaja principal es la centralización de todas las operaciones del restaurante en una única plataforma, lo que mejora la eficiencia y el control. Sin embargo, su complejidad y costo inicial pueden ser elevados, y muchas de sus funcionalidades exceden las necesidades de una operación más simple como la de una cantina universitaria.

El Vacío que Cantina UCC Busca Llenar:

El proyecto Cantina UCC no compite directamente con las plataformas de delivery ni con los software de gestión integral. En cambio, se posiciona como un sistema de pedidos y pagos interno, diseñado específicamente para resolver la ineficiencia del actual proceso presencial y manual de la Universidad Católica de Córdoba, donde los usuarios deben hacer una fila para pagar y otra para seleccionar y retirar su comida.

La solución se enfoca en la recogida rápida de pedidos anticipados por parte de usuarios que ya se encuentran en el campus, llenando el vacío de una solución a medida que evite tanto las altas comisiones de los intermediarios como la complejidad innecesaria de los sistemas de gestión integral.

La eficacia de este enfoque ha sido corroborada en el sector de servicios de alimentación. Por ejemplo, la empresa Gallina Blanca (**anexo 3**) logró automatizar la recepción y gestión de pedidos, lo que resultó en la liberación de aproximadamente 500 horas anuales en su departamento de atención al cliente. Otro caso relevante es el de la Plataforma Integrada de Auto Atención (PIA) para ICB Food Service (**anexo 4**), que al digitalizar y optimizar la gestión de pedidos, mejoró notablemente la eficiencia operativa de la empresa.

Aunque el proyecto Cantina UCC está orientado a una escala menor, estos ejemplos demuestran que la adopción de tecnologías para la gestión de pedidos anticipados reduce significativamente los tiempos de espera, optimiza los recursos y mejora la satisfacción del

cliente. Al crear una solución interna, el proyecto elude los costos y la dependencia de plataformas de terceros, optimiza los márgenes de ganancia y permite un control total sobre la experiencia del usuario, respondiendo de manera inteligente a la necesidad de agilidad en un contexto de recreos cortos y alta afluencia que las soluciones comerciales genéricas no abordan.

La siguiente tabla compara las soluciones digitales existentes, destacando la posición única de Cantina UCC en el caso:

Tipo de Solución	**Características Clave**	**Ventajas Generales**	**Desventajas Generales**	**Relevancia para Cantina UCC**
App de Delivery Externa	Logística de entrega, amplia base de clientes, marketing.	Incremento de ventas, mayor alcance, comodidad para el cliente.	Altas comisiones (hasta 30%), dependencia y control limitado, complejidad multicanal.	No aplica directamente; Cantina UCC no ofrece delivery externo.
Software POS Integral	Gestión de inventario, facturación, nóminas, análisis de ventas, POS.	Centralización de operaciones, control detallado, reportes avanzados.	Costo inicial elevado, complejidad excesiva para una cantina universitaria, funcionalidades no esenciales.	Demasiado complejo y costoso para la necesidad específica; Cantina UCC es más ágil.
Sistema de pedido Interno (a medida)	Pedidos y pagos anticipados, gestión de planes de comida, interfaz cliente/administrador.	Control total sobre la operación, retención de ganancias, experiencia de usuario personalizada.	Requiere desarrollo propio y mantenimiento, sin logística de delivery integrada por defecto.	Solución a medida que llena un vacío al ofrecer control total en un entorno institucional.

4. Tecnologías Investigadas

Para el desarrollo de la aplicación **Cantina UCC**, se evaluaron diversas tecnologías en las capas de frontend, backend, despliegue y servicios en la nube, así como integración y despliegue continuo. La selección final se basó en criterios de rendimiento, escalabilidad, facilidad de desarrollo y adecuación al contexto del proyecto.

Frontend | UI

La capa de interfaz de usuario es crucial para la interacción del usuario y la adaptabilidad a diferentes dispositivos. Se consideraron los siguientes frameworks y librerías:

Next.js:

- **Características:** Next.js es un framework de JavaScript de código abierto construido sobre React, que facilita la creación de aplicaciones y sitios web rápidos y fáciles de usar. Permite el renderizado del lado del servidor (SSR) y la generación de sitios estáticos (SSG), así como la combinación con el renderizado del lado del cliente (CSR), lo que lo convierte en una arquitectura muy potente. Ofrece enrutamiento automático, optimización de imágenes, actualización y recarga rápida, entre otros.

- **Ventajas:** Proporciona un rendimiento muy alto y una gran eficiencia gracias a sus capacidades de SSR y SSG, lo que se traduce en tiempos de carga reducidos y una mejor experiencia de usuario. Es altamente amigable con el SEO, lo cual es fundamental para la visibilidad de una aplicación pública (aunque no es relevante en este caso). Simplifica el desarrollo web con su sistema de enrutamiento automático. Además, incluye optimización automática de imágenes y funcionalidades de actualización rápida.

- **Desventajas:** La curva de aprendizaje puede ser más pronunciada si el desarrollador no tiene conocimientos previos sólidos de React. Aunque su comunidad está creciendo, es más pequeña en comparación con la de React puro. La implementación correcta del SSR puede ser más compleja de configurar de lo que parece.

- **Relevancia para Web App Responsive:** Sus capacidades de renderizado híbrido y optimización automática lo hacen ideal para construir aplicaciones web rápidas y con buen SEO, lo cual es crucial para una interfaz de usuario pública y accesible desde cualquier dispositivo. La optimización de imágenes y la precarga de enlaces contribuyen a una navegación fluida y rápida en cualquier pantalla.

React.js:

- **Características:** React.js es una biblioteca de JavaScript desarrollada por Facebook, centrada exclusivamente en la construcción de interfaces de usuario (UI). Una de sus características clave es el uso de un DOM Virtual, una representación ligera del DOM real, que permite actualizar solo los elementos necesarios en pantalla, mejorando el rendimiento. Facilita la creación de Single Page Applications (SPA) y se basa en una arquitectura de componentes reutilizables, lo que promueve el desarrollo modular y la escalabilidad.

- **Ventajas:** Es relativamente fácil de aprender, especialmente para quienes ya conocen JavaScript, gracias a su sintaxis intuitiva. Ofrece alta flexibilidad y un excelente rendimiento, integrándose bien con otras librerías y respondiendo eficazmente bajo carga. Cuenta con una comunidad de desarrolladores muy amplia y activa, y el respaldo continuo de Facebook. Es ideal para interfaces dinámicas donde los datos cambian frecuentemente.

- **Desventajas:** Al ser solo una biblioteca de UI, React.js requiere la integración con otras herramientas (como React Router para enrutamiento o Redux para gestión de estado) para construir aplicaciones complejas, lo que puede aumentar la complejidad del proyecto. Puede haber una falta de documentación oficial unificada y su excesiva libertad estructural puede llevar a proyectos mal gestionados si no hay

un patrón arquitectónico claro.

- **Relevancia para Web App Responsive:** Su eficiencia con el DOM Virtual y su enfoque basado en componentes son excelentes para crear interfaces de usuario interactivas y rápidas que se adaptan bien a diferentes tamaños de pantalla, garantizando una experiencia de usuario positiva incluso con grandes cantidades de datos dinámicos.

Vue.js:

- **Características:** Vue.js es un framework progresivo de JavaScript para la construcción de interfaces de usuario, reconocido por su simplicidad y flexibilidad. Es notablemente ligero, con un paquete principal comprimido que pesa solo 18 KB. Ha adoptado los mejores conceptos de React y Angular, ofreciendo una combinación equilibrada.

- **Ventajas:** Presenta una curva de aprendizaje más suave en comparación con React y Angular, lo que lo hace accesible incluso para principiantes. Es fácil de entender e integrar en proyectos existentes, permitiendo incluir componentes en aplicaciones ya desarrolladas. Ofrece un excelente rendimiento debido a su tamaño reducido.

- **Desventajas:** Su comunidad, aunque activa, es relativamente más pequeña que la de React o Angular, especialmente fuera de China, lo que puede limitar la disponibilidad de plugins y librerías para requisitos muy específicos. Puede no ser la opción ideal para proyectos de muy gran escala, aunque es utilizado por empresas como IBM y Adobe.

- **Relevancia para Web App Responsive:** Su simplicidad y ligereza lo hacen adecuado para desarrollar aplicaciones web interactivas y dinámicas que funcionan bien en cualquier dispositivo, especialmente cuando se busca agilidad en el desarrollo y una curva de aprendizaje reducida.

Angular:

- **Características:** Angular es un framework de desarrollo de aplicaciones web de una sola página (SPA) completo y estructurado, basado en TypeScript y mantenido por Google. A diferencia de React, que es una biblioteca, Angular proporciona una solución integral que incluye inyección de dependencias y una arquitectura más estructurada, con un compilador Ivy que optimiza el tamaño de las aplicaciones.

- **Ventajas:** Ofrece una solución "todo en uno" con una arquitectura robusta y estructurada, lo que lo hace ideal para proyectos complejos y empresariales que requieren una gran organización y escalabilidad. Cuenta con un fuerte soporte de Google y una evolución constante con múltiples versiones que mejoran el rendimiento y las herramientas de desarrollo.

- **Desventajas:** Su curva de aprendizaje es más empinada en comparación con React y Vue.js, lo que puede hacerlo menos ideal para proyectos con tiempos de desarrollo más cortos o de menor envergadura. Su enfoque más completo puede resultar en una mayor complejidad inicial.

- **Relevancia para Web App Responsive:** Aunque es una herramienta potente para

construir SPAs robustas y escalables, su enfoque más estructurado y completo puede ser excesivo para aplicaciones que priorizan la agilidad y la ligereza en el desarrollo *responsive*, a menos que el proyecto esté destinado a escalar a una complejidad empresarial significativa.

Tabla comparativa de Frameworks Frontend para Aplicaciones Web Responsive

Framework	**Características Clave**	**Ventajas**	**Desventajas**	**Relevancia para App Web Responsive**
Next.js	SSR/SSG, CSR, enrutamiento automático, optimización de imágenes.	Alto rendimiento, amigable con SEO, simplifica el desarrollo, escalable.	Curva de aprendizaje si no se conoce React, comunidad más pequeña.	Ideal para aplicaciones web rápidas y con buen SEO, adaptable a cualquier dispositivo.
React.js	Biblioteca UI, DOM Virtual, componentes reutilizables, SPA.	Fácil de aprender, alta flexibilidad, gran comunidad, ideal para UI dinámicas.	Requiere herramientas adicionales para proyectos complejos, menos amigable con SEO por defecto.	Excelente para interfaces de usuario interactivas y rápidas que se adaptan bien a diferentes tamaños de pantalla.
Vue.js	Progresivo, ligero (18 KB), simplicidad, flexibilidad, reactividad.	Curva de aprendizaje suave, fácil integración, excelente rendimiento.	Comunidad más pequeña (especialmente fuera de China), menos plugins/librerías.	Adecuado para aplicaciones web interactivas y dinámicas que funcionan bien en cualquier dispositivo, agilidad en desarrollo.
Angular	Framework completo SPA, TypeScript, arquitectura estructurada, inyección de dependencias.	Solución "todo en uno", robusto para proyectos complejos, fuerte soporte de Google.	Curva de aprendizaje empinada, puede ser complejo para proyectos pequeños.	Potente para SPAs, pero su estructura puede ser excesiva si se prioriza agilidad y ligereza en el desarrollo responsive.

Backend

Para el desarrollo del *backend*, que gestiona la lógica de negocio, los datos y las interacciones con la base de datos y servicios externos, se consideraron varias opciones:

FastAPI:

- **Características:** FastAPI es un framework web moderno y rápido para construir APIs con Python, basado en estándares abiertos como OpenAPI (anteriormente Swagger) y JSON Schema. Se destaca por su muy alto rendimiento, comparable con NodeJS y Go. Permite una velocidad de programación significativamente mayor (aproximadamente un 200% a 300% más rápido) y reduce los errores inducidos por desarrolladores en un 40%. Ofrece soporte para programación asíncrona, autocompletado en editores y documentación interactiva automática (Swagger UI y ReDoc).

- **Casos de Uso para APIs de Gestión de Pedidos:** Es ideal para construir APIs robustas y eficientes que requieren validación de datos rigurosa (incluso para objetos JSON profundamente anidados), conversión automática de datos de entrada y salida, y una documentación clara y automática. Su alto rendimiento y soporte asíncrono lo hacen particularmente adecuado para sistemas transaccionales con alta demanda, como una API de gestión de pedidos, donde la integridad de los datos y la respuesta rápida son cruciales.

Django:

- **Características:** Django es un framework robusto de Python para el desarrollo de aplicaciones web, conocido por su filosofía de "baterías incluidas". Proporciona un potente ORM (Object-Relational Mapper) que facilita la interacción con la base de datos, un panel de administración integrado, serialización de datos en diferentes formatos (como JSON y XML), y herramientas para autenticación de usuarios. Cuenta con una excelente documentación y una gran comunidad.

- **Casos de Uso para APIs de Gestión de Pedidos:** Es adecuado para aplicaciones complejas que requieren una base de datos bien estructurada y una gran cantidad de funcionalidades pre-construidas. Su ORM simplifica la gestión de productos, órdenes y usuarios en la base de datos, mientras que sus características de autenticación y serialización son valiosas para construir APIs seguras y eficientes para un sistema de pedidos.

Flask:

- **Características:** Flask es un micro-framework de Python, caracterizado por ser ligero y flexible. Permite a los desarrolladores construir APIs RESTful con un alto grado de control sobre la estructura del proyecto. Facilita la implementación de funcionalidades como la autenticación (por ejemplo, con Flask-JWT), paginación y ordenamiento de datos, ofreciendo una sintaxis sencilla para definir rutas con parámetros dinámicos.

- **Casos de Uso para APIs de Gestión de Pedidos:** Es ideal para construir APIs RESTful personalizadas y de tamaño pequeño a mediano, donde se valora la flexibilidad y un control granular sobre cada componente. Su ligereza lo hace adecuado para proyectos que no requieren la complejidad de framework "todo incluido" y que buscan una implementación eficiente de *endpoints* específicos.

Node.js (Express.js):

- **Características:** Node.js es un entorno de ejecución de JavaScript de un solo hilo y multiplataforma, basado en el motor V8 de Google Chrome, ideal para aplicaciones de red escalables y en tiempo real. Express.js es un framework web minimalista y flexible construido sobre Node.js, que simplifica la creación de APIs y aplicaciones web. Ambos permiten la codificación en JavaScript tanto para el *frontend* como para el *backend*, lo que puede agilizar el proceso de desarrollo.

- **Casos de Uso para APIs de Gestión de Pedidos:** Es excelente para construir APIs REST ligeras y rápidas, especialmente en escenarios que requieren comunicación en tiempo real (como chats o *streaming* de datos) o manejo intensivo de operaciones de entrada/salida (E/S). Su modelo asíncrono y no bloqueante lo hace eficiente para gestionar un gran número de conexiones simultáneas, lo cual es ventajoso para una API de gestión de pedidos con potencial alta concurrencia.

Go (Gin Gonic):

- **Características**: Go es un lenguaje de programación compilado y fuertemente tipado creado por Google, diseñado para ser simple, eficiente y confiable. Destaca por su **excelente manejo de la concurrencia** a través de goroutines. Gin Gonic es un framework web minimalista y de alto rendimiento para Go. Proporciona un enrutador muy rápido y una API sencilla para construir servicios web, con un bajo consumo de memoria. Al ser un lenguaje compilado, el despliegue se simplifica a un único archivo binario.
- **Casos de Uso para APIs de Gestión de Pedidos**: Es la opción ideal para microservicios que demandan el **máximo rendimiento posible** y una latencia mínima. Su capacidad para manejar miles de conexiones concurrentes de manera eficiente lo hace perfecto para sistemas de pedidos a gran escala o en tiempo real. La seguridad que provee el tipado estático reduce errores en tiempo de ejecución, algo crítico para sistemas transaccionales.

Tabla comparativa de Frameworks Backend para APIs de Gestión de Pedidos:

Framework	**Características Clave**	**Ventajas**	**Desventajas**	**Casos de Uso Relevantes para APIs de Gestión de Pedidos**
FastAPI	Python, alto rendimiento, asíncrono, OpenAPI/JSON Schema, validación automática.	Muy rápido de programar y ejecutar, reduce errores, documentación interactiva automática.	Ecosistema más joven que Django/Node.js, menos "baterías incluidas".	APIs de alto rendimiento, microservicios, sistemas transaccionales con validación de datos crítica.
Django	Python, "baterías	Desarrollo rápido de	Puede ser "demasiado"	Aplicaciones empresariales

	incluidas", ORM, panel admin, autenticación.	aplicaciones complejas, gran comunidad, robusto.	para APIs simples, menos flexible para microservicios puros.	completas, APIs con lógica de negocio compleja y gestión de datos relacionales.
Flask	Python, micro-framework, ligero, flexible, modular.	Gran control sobre la estructura, ideal para APIs REST personalizadas, rápido para proyectos pequeños.	Requiere más configuración manual para funcionalidades comunes, menos "baterías incluidas".	APIs RESTful ligeras, microservicios específicos, prototipado rápido.
Node.js (Express.js)	JavaScript, entorno de ejecución V8, asíncrono, no bloqueante, minimalista.	Alto rendimiento para E/S, ideal para tiempo real, un solo lenguaje (full-stack JS), gran ecosistema NPM.	Curva de aprendizaje de asincronía, manejo de errores puede ser complejo, un solo hilo puede ser un cuello de botella si no se gestiona bien.	APIs REST rápidas y ligeras, aplicaciones de chat en tiempo real, streaming de datos, microservicios con alta concurrencia.
Go (Gin Gonic)	Go, compilado, alto rendimiento, concurrencia (goroutines), minimalista.	Rendimiento extremadamente alto, bajo consumo de memoria, despliegue simplificado (binario único), fuertemente tipado.	Curva de aprendizaje de Go puede ser mayor, ecosistema de librerías menos extenso que Node.js/Python.	APIs de muy alta performance, microservicios críticos donde la latencia es clave, sistemas con alta concurrencia.

Despliegue y Servicios en la Nube:

La elección de una plataforma de nube es fundamental para la escalabilidad, seguridad y disponibilidad de la aplicación. Se compararon los tres principales proveedores:

AWS (Amazon Web Services):

- **Características:** AWS es el líder del mercado en servicios en la nube, ofreciendo una gama extremadamente amplia y madura de más de 200 servicios. Proporciona una infraestructura altamente escalable y globalmente disponible, con 99 Zonas de Disponibilidad en 31 regiones geográficas.

- **Servicios Clave (relevantes para aplicaciones):** Incluye Elastic Compute Cloud (EC2) para capacidad informática segura y escalable, S3 para almacenamiento de objetos, y servicios de bases de datos como RDS. También ofrece servicios de contenedores compatibles con Docker y Kubernetes.

- **Consideraciones:** A pesar de su robustez y amplitud, AWS puede resultar costoso y complejo para proyectos más pequeños o con presupuestos limitados, debido a su vasta gama de servicios y modelos de precios.

Google Cloud Platform (GCP):

- **Características:** GCP ofrece una infraestructura confiable con un fuerte enfoque en el desarrollo y despliegue de aplicaciones escalables. Una de sus ventajas distintivas es la disponibilidad de una capa gratuita, que puede ser muy útil para proyectos de pequeña escala o en sus fases iniciales. Además, GCP se destaca por su compromiso con la sostenibilidad, operando con energía 100% renovable en todos sus centros de datos.

- **Servicios Clave (relevantes para Cantina UCC):**

    - **Firestore:** Una base de datos NoSQL que permite el almacenamiento y la sincronización de datos en tiempo real, ideal para la gestión dinámica de pedidos.

    - **Cloud Run:** Un servicio que facilita el despliegue de aplicaciones en contenedores con un modelo *serverless*, escalando automáticamente según la demanda, lo que optimiza el uso de recursos y costos.

    - **Cloud Storage (Buckets):** Un servicio de almacenamiento de objetos diseñado para guardar imágenes y otros archivos estáticos de manera eficiente y escalable.

    - **Secrets Manager:** Una herramienta para la gestión segura de secretos y credenciales, esencial para la seguridad de la aplicación.

- **Consideraciones:** Su capa gratuita y el modelo de precios competitivo lo hacen particularmente atractivo para *startups* y proyectos con presupuesto limitado. Su compromiso con la energía renovable añade un valor significativo desde una perspectiva medioambiental.

Microsoft Azure:

- **Características:** Azure es la plataforma de nube de Microsoft, que proporciona herramientas y servicios similares a AWS y GCP. Ocupa el segundo lugar en cuota de mercado y es conocida por su fuerte integración con soluciones empresariales y herramientas de desarrollo de Microsoft.

- **Servicios Clave (relevantes para aplicaciones):** Ofrece Máquinas Virtuales como su principal servicio de computación, Almacenamiento Blob para objetos, y Azure Synapse Analytics para lagos de datos y almacenes. También incluye servicios de computación sin servidor.

- **Consideraciones:** Aunque potente, su interfaz puede ser percibida como más difícil

de usar para nuevos usuarios. Ofrece precios competitivos de pago por uso y flexibilidad para cancelar planes en cualquier momento.

Tabla comparativa de Plataformas de Nube para Despliegue de Aplicaciones:

Plataforma	**Cuota de Mercado**	**Servicios Clave Relevantes (Ejemplos)**	**Ventajas**	**Desventajas**	**Consideraciones Específicas**
AWS	Líder (31%)	EC2 (computación), S3 (almacenamiento), RDS (bases de datos), servicios de contenedores.	Amplitud de servicios, madurez, escalabilidad global, gran ecosistema.	Puede ser costoso y complejo para proyectos pequeños, curva de aprendizaje.	Ideal para sistemas complejos y empresariales, amplia oferta de servicios.
GCP	Tercero (10%)	Firestore (NoSQL), Cloud Run (serverless), Cloud Storage (objetos), Secrets Manager.	Capa gratuita, compromiso con energía 100% renovable, precios competitivos, servicios serverless.	Menor cuota de mercado que AWS/Azure, ecosistema en crecimiento.	Muy útil para startups y proyectos con presupuesto limitado, fuerte enfoque en sostenibilidad.
Microsoft Azure	Segundo (23%)	Máquinas Virtuales (computación), Blob Storage (objetos), Synapse Analytics (data lake), servicios sin servidor.	Fuerte integración con Microsoft, soporte empresarial, amplia gama de servicios.	Interfaz puede ser difícil para nuevos usuarios, percepción de mayor complejidad.	Preferido por empresas con ecosistemas Microsoft, precios competitivos de pago por uso.

Herramientas de Integración y Despliegue Continuo (CI/CD)

Jenkins

Características: Jenkins es una de las herramientas más veteranas y flexibles para CI/CD. Permite automatizar el ciclo de vida completo del software (construcción, pruebas, despliegue). Tiene un ecosistema muy amplio de plugins, lo que lo hace extremadamente configurable.

Servicios Clave / Capacidades:

- Pipelines altamente personalizables.
- Amplia integración con otras herramientas (Docker, Kubernetes, GitHub, Slack, etc.). Despliegue en prácticamente cualquier infraestructura (on-premise o cloud).

Consideraciones:

- Requiere instalación y mantenimiento propios (no es SaaS).
- La curva de aprendizaje puede ser elevada.
- Ideal para equipos con experiencia técnica que buscan control total.

GitHub Actions

Características: GitHub Actions es la solución de CI/CD nativa de GitHub. Permite definir flujos de trabajo en YAML que se disparan en función de eventos (push, pull request, releases, etc.).

Servicios Clave / Capacidades:

- Integración nativa con repositorios de GitHub.
- Amplio marketplace de acciones reutilizables.
- Ejecución en runners de GitHub o runners propios.
- Soporte directo para despliegue en plataformas como AWS, GCP, Azure y Vercel.

Consideraciones:

- Más sencillo de configurar que Jenkins.
- Escalable según el tamaño del proyecto.
- Ideal para startups, proyectos open source y equipos que ya trabajan con GitHub.

GitLab CI/CD

Características: GitLab CI/CD viene integrado en GitLab, lo que lo convierte en una solución todo en uno para repositorios, issues, CI/CD y monitoreo.

Servicios Clave / Capacidades:

- Pipelines definidos en archivos YAML.
- Soporte nativo para contenedores y Kubernetes.
- Integración con gestión de proyectos y control de versiones.
- Permite self-hosted runners o usar los de GitLab.

Consideraciones:

- Ideal si ya se trabaja en GitLab (flujo completo en una sola plataforma).
- Puede ser más cerrado en comparación con Jenkins en cuanto a integraciones externas.
- Su comunidad y ecosistema son más pequeños que los de GitHub.

Tabla resumen

Herramienta	Tipo / Modelo	Ventajas	Desventajas	Consideraciones Específicas
Jenkins	Open source, self-hosted	Extremadamente flexible, ecosistema maduro de plugins, soporta cualquier infraestructura.	Configuración inicial compleja, mantenimiento propio, curva de aprendizaje alta.	Ideal para equipos técnicos que requieren control total y entornos complejos.
GitHub Actions	SaaS integrado en GitHub	Configuración sencilla, integración nativa con GitHub, marketplace de acciones, escalabilidad.	Dependencia de GitHub, runners gratuitos limitados.	Muy útil para proyectos en GitHub, startups y open source.
GitLab CI/CD	SaaS / self-hosted (incluido en GitLab)	Flujo completo en una sola plataforma (repos, issues, CI/CD), integración nativa con Kubernetes.	Menor ecosistema que GitHub, menos flexible que Jenkins.	Ideal para equipos que ya trabajan en GitLab y buscan una solución todo en uno.

Plataformas de Pago en el Ecosistema Argentino (Opciones para integrar pagos online)

Para tomar una decisión informada, se analizaron las principales opciones disponibles en el mercado argentino. La siguiente evaluación se basa en criterios críticos para un proyecto en su fase inicial y de crecimiento, como la estructura de costos, la liquidez, la oferta de pagos y la facilidad de implementación.

Criterios de Evaluación

La comparación se estructura en torno a cinco métricas fundamentales:

1. **Modelo de Comisiones**: Se analiza la estructura de costos, que generalmente incluye un porcentaje variable por transacción, a veces un costo fijo, y el Impuesto al

Valor Agregado (IVA). Las comisiones pueden variar significativamente según el método de pago y el plazo de acreditación de los fondos.

2. **Plazos de Acreditación de Fondos**: Este factor es vital para la gestión del flujo de caja de un negocio. Los plazos pueden ir desde la acreditación inmediata, con una comisión más alta, hasta 30 días o más para obtener una tasa más competitiva.

3. **Métodos de Pago Aceptados**: Se evalúa la amplitud de opciones que la pasarela ofrece al cliente final, incluyendo tarjetas de crédito y débito y saldo de billeteras digitales.

4. **Facilidad de Integración**: Se considera la disponibilidad y calidad de la documentación técnica, APIs, y Kits de Desarrollo de Software (SDKs),lo cual impacta directamente en los costos y tiempos de desarrollo.

5. **Mercado Objetivo y Soporte**: Cada pasarela suele estar optimizada para un segmento de negocio específico, desde pequeños emprendedores hasta grandes corporaciones. La calidad y disponibilidad del soporte técnico es también un diferenciador clave para resolver incidencias de manera eficiente.

Matriz Comparativa de Pasarelas de Pago en Argentina

Característica	**Mercado Pago**	**PayU**	**Ualá Bis**	**Mobbex**	**Payway (Prisma)**
Comisiones (aprox.)	Variable: 1.99% a 5.99% + IVA (depende del plazo)	3.49% + fijo + IVA	4.4% + IVA (crédito), 2.9% + IVA (débito)	~4% + IVA (plan simple)	Negociación directa con bancos + fee de la pasarela
Plazos Acreditación	Inmediato, 14 o 30 días	Variable, con retiros mensuales limitados	Inmediato	5-12 días hábiles	Directo en cuenta bancaria (rápido)
Métodos de Pago	Todos (Tarjetas, Efectivo, QR, Saldo en cuenta)	Amplia variedad, foco regional	Tarjetas, Link de pago	Tarjetas, Transferencias, Cripto (vía Binance)	Tarjetas, QR, Débito automático
Integración	Excelente (APIs, SDKs, Plugins)	Buena (APIs, SDKs)	Sencilla, foco en links de pago	Buena, foco en e-commerce y recurrencia	Compleja, requiere desarrollo a medida y convenios bancarios

Mercado Objetivo	Emprendedores a grandes empresas	Pymes y corporaciones con operación en LatAm	Emprendedores y pequeños comercios	Pymes y negocios con pagos recurrentes	Grandes empresas con alto volumen
Soporte	24/7 (aunque a veces genérico)	Estándar	Bueno para su nicho	Destacado por su atención personalizada	Soporte corporativo

Infraestructura para la Comunicación Transaccional por Correo Electrónico

Los correos electrónicos transaccionales son mensajes automatizados que se envían a un usuario individual en respuesta a una acción específica realizada por este en una plataforma o aplicación. Ejemplos comunes incluyen confirmaciones de compra, notificaciones de envío, correos para restablecer contraseñas, envío de facturas o confirmaciones de registro. A diferencia del email marketing, que se envía a listas de suscriptores con fines promocionales, los correos transaccionales son funcionales y esperados por el usuario.

Debido a su naturaleza, estos correos tienen tasas de apertura y de clics (CTR) extremadamente altas en comparación con las campañas de marketing. Son un punto de contacto fundamental en el ciclo de vida del cliente y desempeñan un papel crucial en la construcción de confianza. Un correo de confirmación de pedido que llega de forma instantánea y con un formato profesional reafirma al cliente que su compra fue exitosa y que la empresa es fiable.

Impacto en la Operativa

Una gestión deficiente de los correos transaccionales puede tener consecuencias negativas severas. Si un correo de confirmación no llega, llega con retraso o es filtrado a la carpeta de spam, el usuario puede pensar que su transacción falló, lo que genera ansiedad y desconfianza. Esto, a su vez, provoca un aumento en las consultas al equipo de soporte al cliente, incrementando la carga de trabajo y los costos operativos. Por lo tanto, garantizar una alta tasa de entregabilidad (la capacidad de que los correos lleguen a la bandeja de entrada) es una prioridad técnica y de negocio.

Por esto, se evaluaron diferentes enfoques para el envío de correos, desde servicios de API especializados hasta una implementación propia utilizando la librería smtplib de Python con un servidor de Gmail.

Característica	**Servicios Dedicados (SendGrid, Mailgun, Resend)**	**Amazon SES (Simple Email Service)**	**Desarrollo Propio (Gmail + smtplib)**

Modelo de Precios	Planes escalonados con niveles gratuitos funcionales para iniciar.	Modelo de pago por uso, extremadamente económico a gran escala ($0.10 usd por 1,000 correos).	"Gratis" dentro de los límites de la cuenta de Gmail.
Facilidad de Uso	Muy alta. Interfaces amigables, excelente documentación y SDKs para múltiples lenguajes que simplifican la integración.	Baja. Interfaz técnica y compleja, requiere conocimientos del ecosistema AWS para una configuración correcta.	Engañosamente simple para un prototipo, pero muy complejo de gestionar y escalar en un entorno de producción.
Entregabilidad y Analíticas	Excelente. Ofrecen herramientas avanzadas para maximizar la entregabilidad, gestionar la reputación del remitente y analizar métricas detalladas (aperturas, clics, rebotes).	Muy alta, pero depende de una configuración manual correcta de protocolos como SPF y DKIM. Las analíticas son básicas y requieren integración con otros servicios de AWS.	Pobre. No ofrece analíticas. La entregabilidad depende enteramente de la reputación de una cuenta personal, con alto riesgo de ser marcada como spam.
Escalabilidad y Límites	Alta. Diseñados para manejar grandes volúmenes de envío sin límites diarios restrictivos en los planes de pago.	Muy alta. Construido sobre la infraestructura de AWS para escalar a millones de correos.	Muy baja. Límites de envío diarios estrictos (500 para cuentas gratuitas, 2,000 para Workspace) que pueden bloquear la cuenta por hasta 24 horas si se exceden.

Ideal Para...	Proyectos que buscan una solución robusta, fácil de implementar y con analíticas potentes desde el inicio.	Proyectos que priorizan el costo por encima de todo y ya operan dentro del ecosistema de AWS.	Pruebas, prototipos o proyectos personales de muy bajo volumen. No es una opción viable para una aplicación en producción.

Estrategias de Pruebas del Sistema

Para garantizar la calidad, fiabilidad y correcto funcionamiento de la aplicación, es fundamental implementar una estrategia de pruebas multifacética que abarque diferentes niveles del sistema. Esta estrategia se centra en la detección temprana de errores, la validación de la lógica de negocio y la confirmación de que el software cumple con las expectativas de los usuarios finales. Las principales etapas de prueba incluyen las pruebas unitarias, las pruebas de integración y las pruebas de aceptación de usuario (UAT).

Pruebas Unitarias

Las pruebas unitarias constituyen la primera línea de defensa contra los errores de software. Su objetivo es verificar el correcto funcionamiento de las unidades de código más pequeñas y aisladas, como funciones, métodos o componentes individuales.

- **Objetivo**: Asegurar que cada pieza del código se comporte como se espera de forma independiente. Por ejemplo, se probaría una función que calcula el total de un pedido, una que valida el formato de un correo electrónico o un componente de la interfaz que muestra el precio de un producto.
- **Metodología**: Estas pruebas son automatizadas y se ejecutan frecuentemente durante el desarrollo. Al aislar los componentes, se utilizan "mocks" o dobles de prueba para simular dependencias externas (como bases de datos o APIs de terceros), permitiendo que la prueba se centre exclusivamente en la lógica de la unidad bajo análisis.
- **Beneficios**:
    - **Detección Temprana de Errores**: Identifican problemas en las etapas iniciales, reduciendo el costo de su corrección.
    - **Facilitan la Refactorización**: Proporcionan una red de seguridad para modificar y mejorar el código sin introducir regresiones.
    - **Documentación Viva**: Sirven como una forma de documentación técnica del comportamiento esperado de cada unidad.

Pruebas de Integración

Una vez que las unidades individuales han sido validadas, las pruebas de integración se

encargan de verificar que estas interactúan correctamente entre sí.

- **Objetivo**: Detectar fallos en las interfaces y en la comunicación entre diferentes módulos del sistema. En el contexto de Cantina UCC, un ejemplo clave sería probar el flujo completo de un pedido: desde que se añade un producto al carrito (frontend), se procesa el pago (API externa) y se registra el pedido en la base de datos (backend).
- **Metodología**: Implican la combinación de varios módulos para simular un flujo de trabajo real, interactuando con servicios reales o sus versiones de prueba (ej. base de datos de prueba, entorno "sandbox" de una pasarela de pago).
- **Beneficios**:
    - **Validación de Flujos de Datos**: Aseguran que los datos se transmiten correctamente entre componentes.
    - **Detección de Errores de Interfaz**: Identifican problemas de comunicación entre módulos.
    - **Confianza en la Arquitectura**: Verifican que los componentes colaboran de manera efectiva.

Pruebas de Aceptación de Usuario (UAT)

En este tipo de pruebas el software es evaluado por los **usuarios finales** para confirmar que cumple con sus necesidades y con los requisitos del negocio.

- **Objetivo**: Validar que el sistema es "apto para su propósito" desde la perspectiva del usuario. No se centra en encontrar errores de código, sino en verificar que la aplicación resuelve el problema original de una manera intuitiva y eficiente.
- **Metodología**: Un grupo representativo de usuarios finales realiza tareas específicas en un entorno similar al de producción para recopilar feedback sobre la usabilidad y satisfacción general.
- **Beneficios**:
    - **Validación del Negocio**: Confirma que el software entrega el valor esperado.
    - **Garantía de Usabilidad**: Asegura que la aplicación es fácil de usar para su público objetivo.
    - **Reducción de Riesgos**: Minimiza el riesgo de que el producto sea rechazado por los usuarios tras su lanzamiento.

Pruebas Funcionales

Este tipo de prueba valida los requisitos del software desde una perspectiva funcional, simulando escenarios de uso reales para verificar que el sistema se comporta como se espera. Abarcan flujos de trabajo completos de principio a fin (End-to-End) y, a diferencia de las UAT, son ejecutadas típicamente por el equipo de desarrollo o de calidad (QA).

- **Objetivo**: Validar que el sistema cumple con los requisitos funcionales descritos en

las especificaciones. Se centra en probar los flujos de trabajo completos para asegurar que cada funcionalidad opera correctamente desde la perspectiva del caso de uso.

- **Metodología**: Simulación de escenarios de uso reales, ejecutados por el equipo de desarrollo o calidad (QA). No se enfoca en la usabilidad o satisfacción del usuario final, sino en el cumplimiento estricto de los requerimientos funcionales.
- **Beneficios**:
    - **Garantía de Calidad**: Asegura que el software entregado cumple con las especificaciones funcionales.
    - **Validación de Flujos Críticos**: Verifica que los procesos de negocio más importantes funcionan correctamente de principio a fin.
    - **Reducción de Errores Post-Lanzamiento**: Detecta fallos funcionales antes de que el software llegue a los usuarios finales.

PROPUESTA DE SOLUCIÓN

1 Alcance Funcional

Para establecer los requerimientos del sistema, se definirá mediante **historias de usuario**. Las historias de usuario permitirán identificar las funcionalidades clave del sistema.

Este enfoque asegura que el equipo pueda comprender las expectativas del usuario final, priorizando las características más importantes y evitando desviaciones o tareas no esenciales. Las historias de usuario también permitirán una planificación ágil y una evaluación continua del progreso del proyecto.

Historias de usuario:

Usuarios Invitados (sin cuenta)

1. Como usuario invitado, quiero explorar el catálogo de productos.
2. Como usuario invitado, quiero poder agregar productos al carrito, para poder ir armando mi pedido.

Usuario Registrado (extiende usuario invitado)

3. Como usuario registrado, quiero poder iniciar sesión utilizando mi correo electrónico o cuenta de Google, para acceder a mi perfil y personalizar mi experiencia.
4. Como usuario registrado quiero poder realizar una compra y poner un horario para retirarlo, para una mejor experiencia y poder planificar mis pedidos.
5. Como usuario registrado, quiero recibir una confirmación por correo electrónico al finalizar una compra, con las instrucciones para retirar el pedido y la información de la transacción.
6. Como usuario registrado quiero poder comprar planes de comida por varios días en la cantina, para recibir descuentos.
7. Como usuario registrado quiero poder aplicar un plan de comida en una compra.
8. Como usuario registrado, quiero ver el historial de mis compras anteriores, para poder ver la información de todos mis pedidos y mis planes de comida.

Administradores de la cantina:

9. Como administrador de la cantina, quiero poder agregar, editar o eliminar productos desde una interfaz de administración, para mantener actualizado el catálogo de la cantina que ven los usuarios.
10. Como administrador de la cantina quiero poder visualizar los pedidos realizados por los usuarios para poder ver su estado y administrarlos.

11. Como administrador de la cantina quiero poder obtener un informe diario y mensual de las compras realizadas en dichos periodos y los montos de dinero para poder hacer un registro de movimientos.

12. Como administrador quiero poder imprimir los tickets de los pedidos que me van llegando para poder procesarlos en la cocina.

13. Como administrador quiero tener una pantalla con las órdenes que son para la cocina para evitar tener que imprimir el ticket

14. Como administrador quiero poder crear y gestionar los planes de comida para que los usuarios puedan comprarlos

Lo que está incluido en el Alcance Funcional:

- Integración con Mercado Pago: Se integrará la plataforma de pagos Mercado Pago para procesar las compras de los usuarios y permitir pagos seguros.

- Notificaciones por correo electrónico: Los usuarios recibirán notificaciones automáticas relacionadas con el estado de sus pedidos y transacciones.

Lo que queda fuera del Alcance Funcional:

- Gestión de inventarios físicos: El sistema no gestionará el inventario físico de la cantina, solo el catálogo de productos dentro de la plataforma.

- Aplicación móvil: Actualmente, solo se desarrollará una versión web de la aplicación; no se considera una aplicación móvil en esta ni en ninguna fase del proyecto.

- Integración con otros métodos de pago: En esta fase, el sistema se integrará únicamente con Mercadopago. La plataforma incluye la opción de pagar con tarjetas de crédito o débito fuera de la aplicación (por lo que no es requisito excluyente tener una cuenta de Mercadopago) pero todos los pagos serán manejados únicamente a través de este proveedor. Otros proveedores de pago no están contemplados en este alcance.

- Funciones de análisis avanzado: El sistema no incluirá análisis o reportes complejos sobre ventas, productos o usuarios en esta fase inicial.

2 Diseño

Pantallas

<u>En el frontend de la cantina (para clientes)</u>

1. Menú de la cantina: Página principal, donde el cliente puede navegar por el catálogo de productos, buscar por nombre o descripción y filtrar por categoría. Los productos pueden añadirse al carrito.

2. Carrito: Página que muestra todos los productos añadidos al carrito, permitiendo modificar la cantidad de unidades o eliminar productos, permite la selección de un horario para el retiro y también genera el link para comprar a través de Mercadopago

3. Mis ordenes: Página donde se muestran todas las compras realizadas, con la opción de filtrar por fecha y estado de la orden.

4. Cuenta: Página que muestra la información de la cuenta y permite cerrar sesión.

5. Auth: pagina donde iniciar sesión o crear una cuenta.

6. Impacto RSU: Cuenta informativa sobre la Responsabilidad Social Universitaria

7. Gracias x tu compra: página informativa a la que redirige Mercadopago cuando el pago fue exitoso

8. Planes de comida: pagina donde podes armar un plan de comida y comprar el mismo

9. Mis planes de comida: Pagina donde visualizas los planes de comida activos tuyos

En el frontend de administración (para los administradores de la cantina)

1. Productos: Página que muestra una tabla con todos los productos, permitiendo buscar por nombre o descripción y filtrar en orden ascendente o descendente por distintas categorías.

2. Agregar Producto: Página con el formulario para agregar un nuevo producto.

3. Editar Producto: Página con un formulario para editar la información de un producto.

4. Órdenes: Página donde se pueden ver todas las órdenes recibidas, filtrar por fecha, por estado y buscar por id de la orden.

5. Resúmenes: Página donde se pueden visualizar los resumes de ventas diarios y mensuales.

6. Cocina: Página donde se visualizan las órdenes que son enviadas a la cocina para procesar allí.

7. Planes de comida: pagina para crear y administrar los plane de comida que se muestran en la cantina

8. Log in: Página donde iniciar sesión.

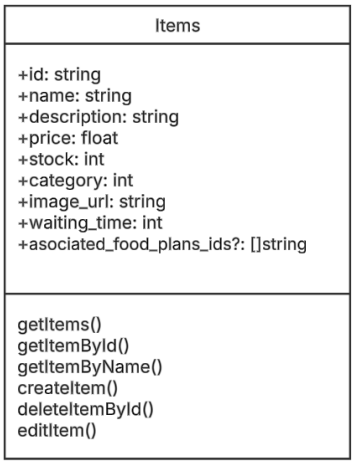
Diagramas

En base de datos tendremos 8 tablas:

1. **Items** (productos):

La tabla **Items** representa el **catálogo de productos o artículos disponibles** en el sistema. Cada instancia de **Items** corresponde a un producto único que puede ser ofrecido para la venta o formar parte de una orden.

Esta entidad almacena todas las características esenciales que definen un producto, permitiendo su identificación, descripción, gestión de inventario y asociación con otros elementos del negocio.



2. **PreOrder**:

La tabla PreOrdenes actúa como un repositorio temporal para la información de una orden antes de que se complete el proceso de pago. Su propósito principal es almacenar todos los detalles necesarios de una compra mientras esta se encuentra en un estado preliminar o pendiente de confirmación.

Una vez que el pago de la orden es exitosamente confirmado, los datos contenidos en la PreOrden se utilizan para generar un objeto Orden definitivo. Este nuevo objeto Orden se almacena de forma persistente, y consecuentemente, la PreOrden correspondiente es eliminada de la base de datos. Esto asegura que PreOrdenes mantenga únicamente registros activos de transacciones aún no finalizadas. La tabla PreOrdenes se somete a una limpieza diaria nocturna, eliminando las pre-órdenes que no fueron pagadas en el día.

Estructura y Composición de PreOrdenes:

Es fundamental comprender que PreOrdenes integra la información de los conceptos Cart (Carrito) y Line (Línea de Artículo) directamente dentro de su estructura. Cart y Line no son tablas separadas en la base de datos, sino que representan la organización interna de los datos que PreOrdenes contiene.

Dentro de cada PreOrden, se almacena un **Cart** que, a su vez, contiene una colección de **Lines**. Cada Line detalla un artículo específico incluido en la pre-orden.

Particularidad del Atributo item en Line:

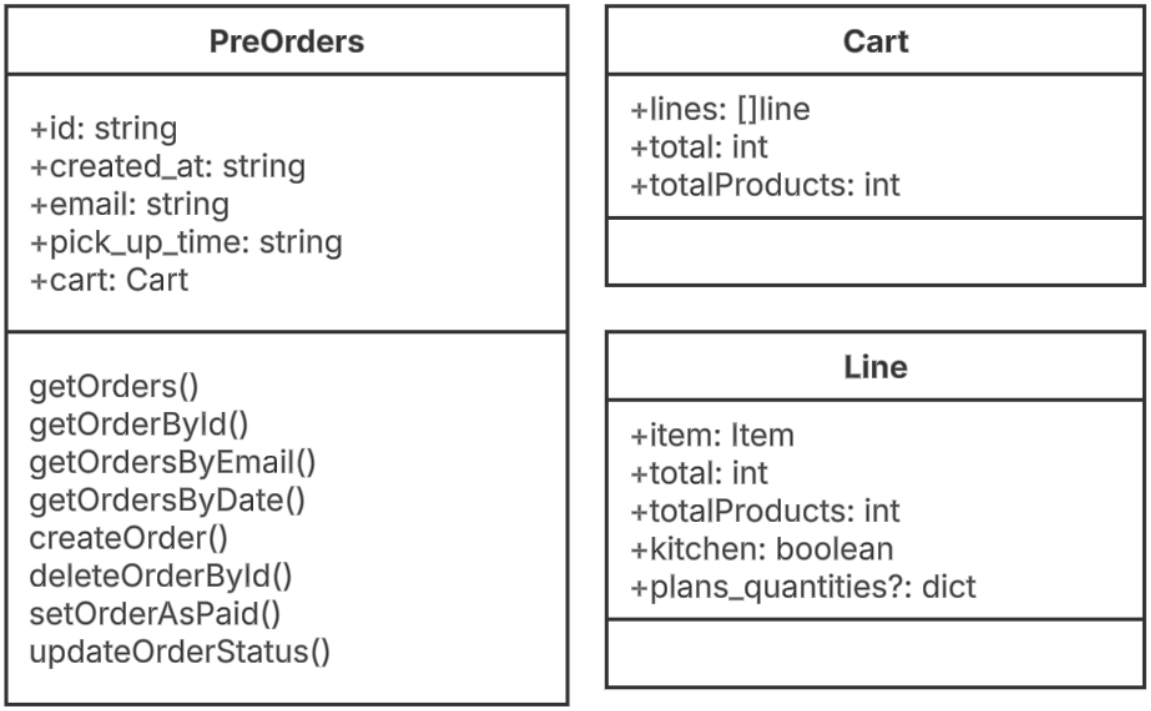
Una característica clave del atributo item dentro de la estructura Line es que no almacena un item_id (identificador de artículo) sino una "fotografía" o una copia completa del artículo en el momento de la pre-orden.

Esta aproximación es crucial por las siguientes razones:

- Preservación de la Integridad: Si el artículo original (en la tabla items) fuera modificado o eliminado después de que se creó la PreOrden, la PreOrden (y

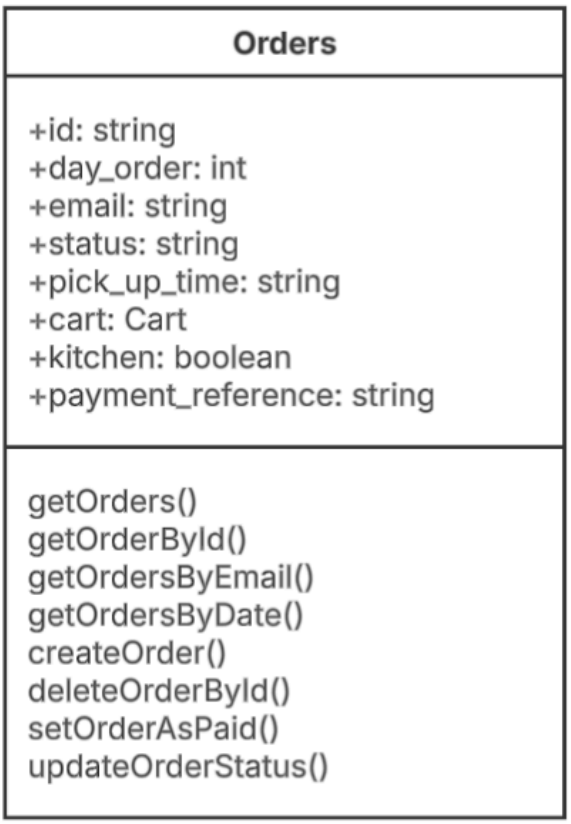
posteriormente la Orden final) mantendría una referencia precisa y completa del artículo tal como era cuando se compró.

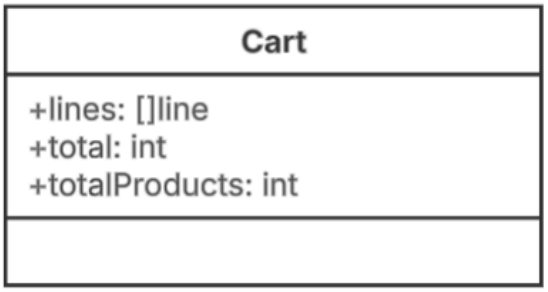
- Independencia de Cambios Futuros: Evita la pérdida de información o referencias rotas que podrían ocurrir si la PreOrden dependiera de un item_id que podría dejar de existir o apuntar a datos modificados Por eso permite tener un registro exacto de las características del producto (precio, descripción, etc.) en el momento de la compra, lo cual es vital para fines de auditoría, devoluciones o seguimiento.

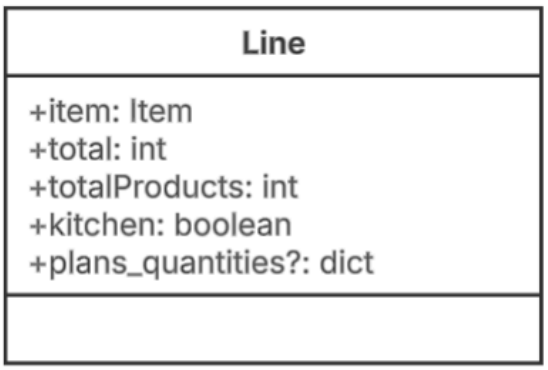


3. **Orders** (órdenes):

La tabla Orders representa el registro persistente y definitivo de las órdenes una vez que el pago ha sido exitosamente confirmado. A diferencia de PreOrders, que es un estado transitorio, cada entrada en Orders es una transacción de compra completada y validada.

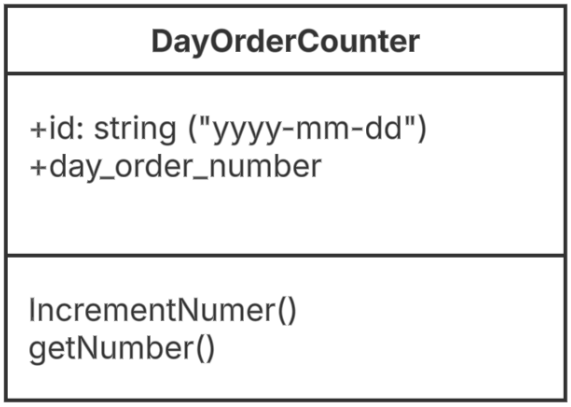






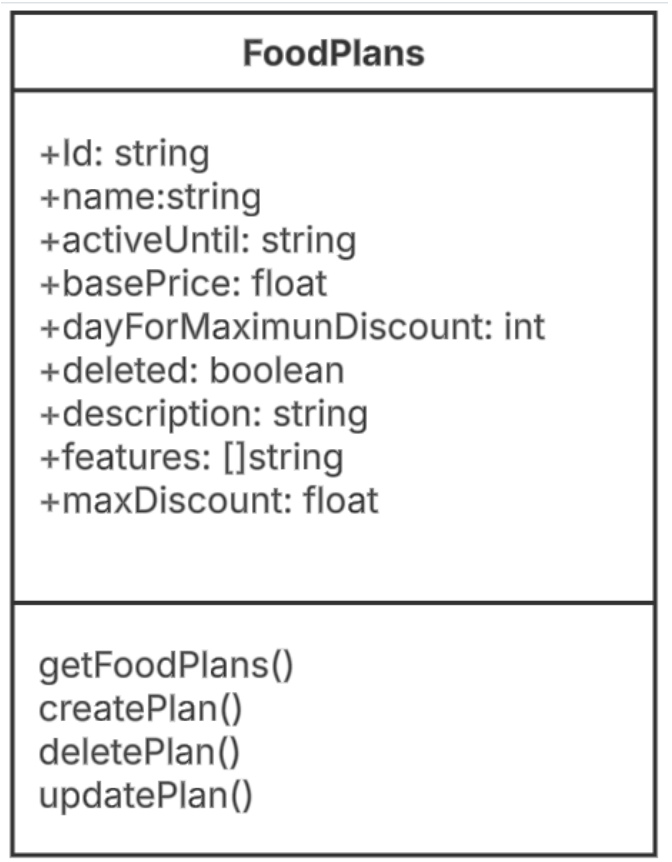
4. **DayOrderCounter**:

Es una tabla o entidad diseñada para llevar un **contador diario de órdenes**. Su objetivo principal es generar números de orden secuenciales para cada día, facilitando así un control organizado de las órdenes para una fecha específica.



5. **FoodPlans**:

Se encarga de definir y administrar los diferentes planes de comida que la Cantina podria ofrecer a los clientes.

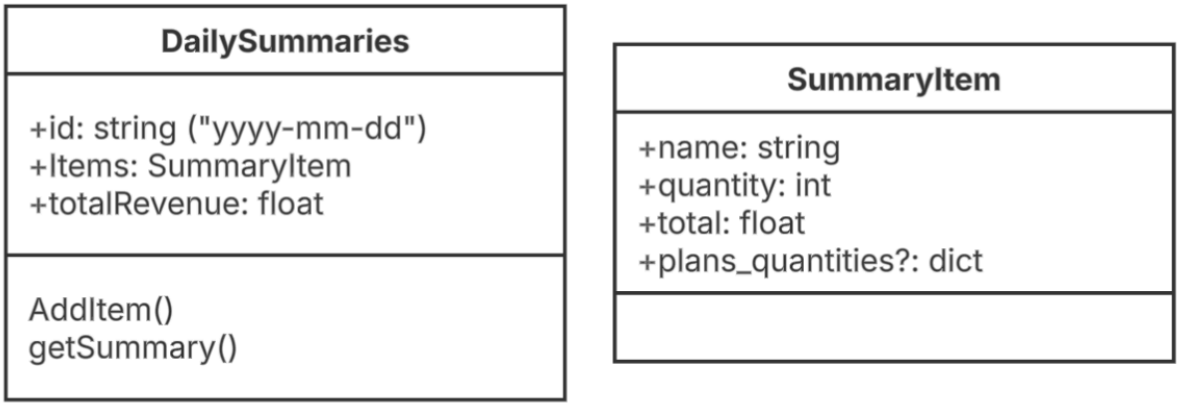


6. **DailySummaries**:

Está diseñada para almacenar y presentar un consolidado de la actividad de ventas de cada día. Su propósito es ofrecer una vista agregada y rápida del rendimiento económico diario del negocio, sin la necesidad de consultar y procesar la gran cantidad de órdenes individuales. Contiene la información resumida de los artículos vendidos y el ingreso total generado en una jornada específica.

Esta tabla se identifica unívocamente por un id que corresponde a la fecha del resumen en formato "aaaa-mm-dd". Contiene la información resumida de los artículos vendidos y el ingreso total generado en una jornada específica.

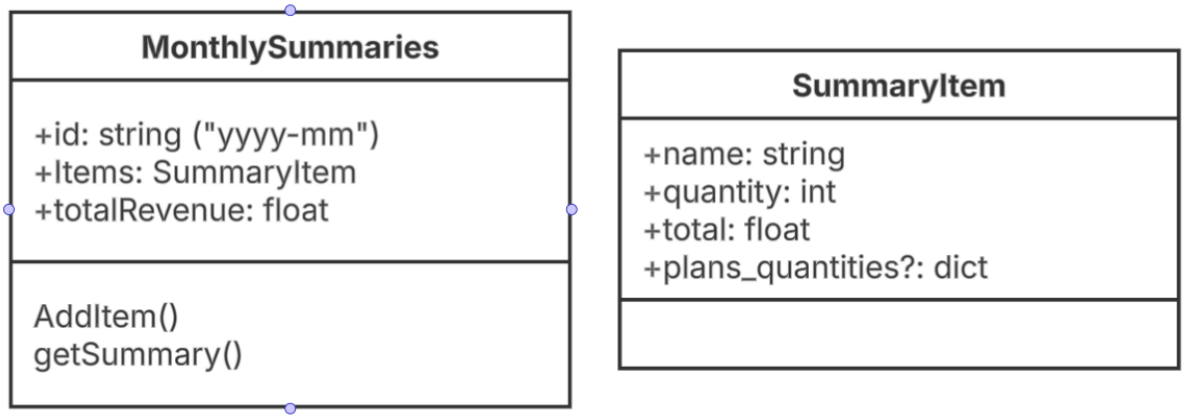
SummaryItem es un componente anidado dentro de DailySummaries, no una tabla en sí. Su función es detallar la información agregada de un artículo específico dentro de un resumen diario. En lugar de mostrar cada instancia individual de un artículo vendido, SummaryItem consolida la cantidad total de unidades de ese artículo que se vendieron y el ingreso total que generaron. Además, puede indicar cómo esas ventas se distribuyeron entre los diferentes planes de comida, si es aplicable.



7. **MonthlySummaries**:

Cumple una función idéntica a **DailySummaries**, pero a una escala temporal diferente: consolida la actividad de ventas a nivel mensual.

Su principal distinción es el id, que identifica cada resumen por el mes en formato "aaaa-mm". Al igual que los resúmenes diarios, contiene la información agregada de SummaryItems y el ingreso total, pero agrupando los datos de todo un mes.



8. OrdersForPlans:

Se dedica a registrar las compras o suscripciones de los planes de comida (**FoodPlans**) por parte de los usuarios. Representa la instancia en que un cliente adquiere un determinado **FoodPlan**, detallando no solo qué plan compró, sino también cuántas unidades o qué cantidad de ese plan adquirió.

Esta tabla es crucial para gestionar el consumo y la vigencia de los planes que los usuarios han prepagado o suscrito. Permite al sistema hacer un seguimiento de cuánto de un plan ha sido utilizado y cuánto queda disponible.

Su id combina el identificador del plan y el email del usuario, asegurando una referencia única para cada suscripción de plan.

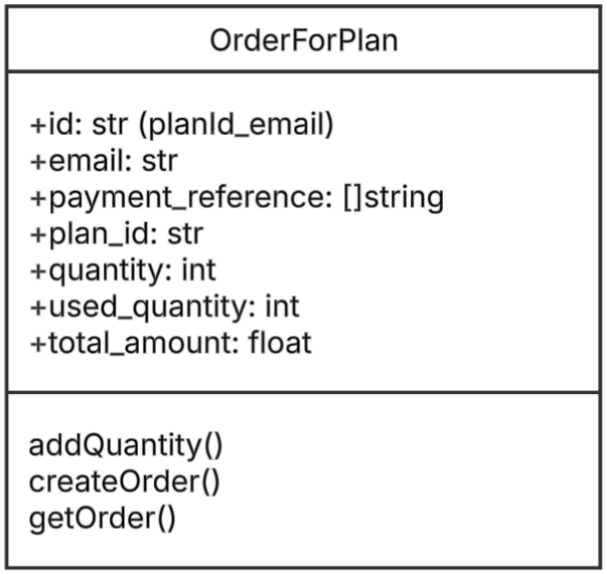
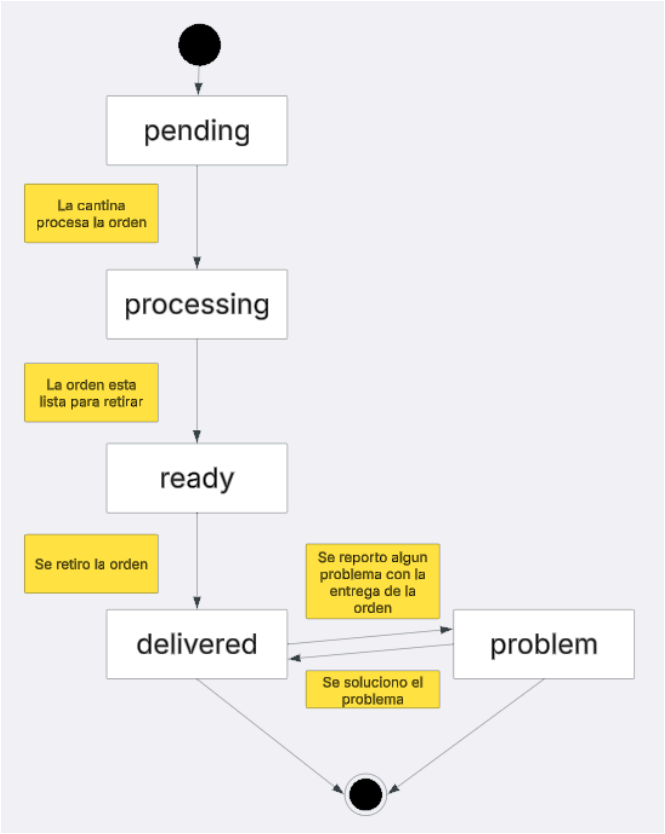


Diagrama de estado de los pedidos:



NOTA: El proceso de gestión de órdenes seguirá un flujo estructurado y solo se aplicará a órdenes que han sido **pagadas**. Tanto la cantina como los usuarios solo pueden ver las órdenes que han sido confirmadas mediante pago; si una orden no ha sido pagada, no se visualizará en ninguna de las interfaces.

Tecnologías elegidas

Frontend: Next.js

Next.js es un framework de React que permite la creación de aplicaciones web con renderizado del lado del servidor y generación de sitios estáticos. Sus principales características incluyen:

- Renderizado Híbrido: Combina el renderizado del lado del servidor y la generación de sitios estáticos, mejorando el rendimiento y la experiencia del usuario.
- Optimización Automática: Realiza división de código y carga optimizada, reduciendo los tiempos de carga.

Backend: FastAPI con Python

FastAPI es un framework web moderno y rápido para la construcción de APIs con Python, basado en estándares como OpenAPI y JSON Schema. Sus ventajas son:

- Alto Rendimiento: Diseñado para ser rápido y eficiente en la construcción de APIs.
- Facilidad de Uso: Ofrece una experiencia de desarrollo sencilla y documentación automática.
- Soporte para Asincronía: Permite la programación asíncrona, mejorando la escalabilidad.

Despliegue y Servicios en la Nube: Google Cloud Platform (GCP)

La elección de GCP para el despliegue de **Cantina UCC** no fue solo una decisión técnica, sino una estrategia multifacética que equilibra la escalabilidad y el rendimiento con la viabilidad económica. La disponibilidad de una capa gratuita en GCP fue un factor decisivo para reducir la barrera de entrada y los costos iniciales del proyecto.

Los servicios específicos incluyen:

- Firestore: Base de datos NoSQL que permite el almacenamiento y sincronización de datos en tiempo real.
- Cloud Run: Servicio que facilita el despliegue de aplicaciones en contenedores, escalando automáticamente según la demanda.
- Buckets: Almacenamiento de objetos para guardar imágenes y otros archivos estáticos.
- Secrets Manager: Gestión segura de credenciales y secretos necesarios para la aplicación.

Esta elección de servicios permite una infraestructura escalable y segura, adaptándose a las necesidades del proyecto.

Integración y Despliegue Continuo: GitHub Actions y Docker Hub

Para garantizar un flujo de trabajo eficiente y automatizado, se implementarán las siguientes herramientas:

- GitHub Actions: Automatiza los procesos de construcción, prueba y despliegue de la aplicación, asegurando que cada cambio en el código se refleje de manera consistente en el entorno de producción.

- Docker Hub: Almacena y gestiona las imágenes de contenedores utilizadas en el despliegue, facilitando la portabilidad y escalabilidad de la aplicación.

Plataforma de pagos

Tras el análisis comparativo, se concluyó que Mercado Pago es la opción estratégica más sólida para el proyecto. Esta elección no se basa en ser la alternativa de menor costo por transacción, sino en su capacidad para maximizar la probabilidad de éxito comercial a través de un conjunto de ventajas cualitativas y cuantitativas.

La decisión de qué pasarela de pago utilizar trasciende un simple cálculo de comisiones; es una decisión fundamental de experiencia de usuario y marketing. Aunque algunas alternativas como Ualá Bis o Mobbex pueden ofrecer tasas marginalmente inferiores en ciertos escenarios, Mercado Pago ostenta una posición de dominio y reconocimiento en el mercado argentino que ninguna otra plataforma puede igualar.

Este reconocimiento se traduce directamente en una mayor tasa de conversión. Un usuario que llega al checkout y se encuentra con la opción de pagar a través de Mercado Pago se siente en un entorno familiar y seguro. La plataforma ofrece protección tanto al comprador como al vendedor, un factor que disminuye la ansiedad asociada a las compras en línea. Optar por una pasarela menos conocida para ahorrar una fracción porcentual en comisiones podría resultar en un costo mucho mayor derivado del aumento en la tasa de abandono de carritos. Por lo tanto, la comisión de Mercado Pago puede ser interpretada no como un gasto, sino como una inversión estratégica en la confianza del cliente y, en última instancia, en la maximización de las ventas.

Además, Mercadopago resuelve de manera eficiente uno de los mayores desafíos del e-commerce en Argentina: la diversidad de métodos de pago. Con una sola integración, el proyecto puede ofrecer a sus clientes la posibilidad de pagar con tarjetas de crédito (incluyendo planes de cuotas), tarjetas de débito, redes de pago en efectivo (Rapipago/Pago Fácil), transferencias bancarias y, de manera crucial, el saldo disponible en la cuenta de Mercado Pago, una de las billeteras virtuales más utilizadas en el país. Ofrecer esta gama completa de opciones desde el primer día sin la necesidad de gestionar múltiples integraciones y contratos es una ventaja operativa y competitiva decisiva.

Correos transaccionales

Para la infraestructura de envío de emails transaccionales se decidió implementar **Amazon Simple Email Service (SES)** como proveedor de envío de correos. La elección se fundamenta en varias razones:

1. Costo altamente competitivo: SES opera bajo un modelo de pago por uso, con un precio de $0.10 USD por cada 1,000 correos enviados, lo cual lo hace

extremadamente económico a gran escala, especialmente en comparación con proveedores dedicados como SendGrid o Mailgun.

2. Escalabilidad y confiabilidad: Al estar construido sobre la infraestructura de AWS, SES puede manejar millones de correos con una tasa de disponibilidad muy alta, sin limitaciones diarias restrictivas. Esto lo convierte en una solución preparada para el crecimiento del proyecto.

3. Entregabilidad: Aunque requiere una correcta configuración de protocolos como SPF, DKIM y DMARC, una vez realizada, la tasa de entregabilidad es muy alta, minimizando la posibilidad de que los mensajes lleguen a la carpeta de spam.

4. Integración en el ecosistema AWS: El proyecto utilizara Google Cloud para backend y despliegues, pero optar por SES permite mantener la flexibilidad de trabajar con la infraestructura de AWS para un servicio tan crítico como la mensajería.

Para garantizar la profesionalidad y la entregabilidad de los correos, se adquirió el dominio **cantinaucc.com** a través de **Namecheap** (por $15.000 por todo un año). Posteriormente, se realizó la configuración necesaria en los **DNS records** para habilitar el envío de correos desde este dominio mediante SES

Con esta configuración, los correos transaccionales enviados desde **…@cantinaucc.com** tienen una altísima probabilidad de llegar correctamente a la bandeja de entrada de los usuarios, evitando problemas de spam o falsos positivos.

Además de servir como remitente confiable para la mensajería transaccional, el dominio **cantinaucc.com** se utilizará para la parte web del proyecto asi el dominio permitirá direccionar a los usuarios hacia las dos interfaces de la aplicación (la de clientes (cantinaucc.com) y la de administradores (admin.cantinaucc.com)), asegurando una experiencia de navegación profesional y unificada bajo una misma identidad digital.

Algunas aclaraciones

La elección de las distintas tecnologías e integraciones se basa también en la experiencia y conocimientos del equipo de desarrollo. Estas tecnologías ofrecen una combinación de rendimiento, escalabilidad y facilidad de uso que se ajusta a los objetivos del proyecto. Aunque existen otras alternativas, la familiaridad del equipo con las tecnologías elegidas reduce los tiempos de desarrollo y mejora la calidad del producto final.

Arquitectura

La arquitectura de la aplicación **Cantina UCC** se basa en un modelo **cliente-servidor** con un enfoque en servicios en la nube, optimizando el rendimiento, escalabilidad y seguridad. La aplicación está diseñada para ser modular y escalable, con una separación clara entre los distintos componentes que interactúan entre sí.

1. Arquitectura General

El sistema está organizado en componentes independientes que interactúan entre sí a través de interfaces bien definidas. Cada componente cumple un rol específico dentro del

flujo general de la aplicación, lo que favorece la flexibilidad y la capacidad de escalar según la demanda.

Los principales componentes son:

- **Frontend (Cliente):** Gestiona la interacción con los usuarios finales y envía solicitudes al backend.

- **Frontend (Administrador):** Interfaz separada, destinada a la gestión operativa por parte del personal administrativo.

- **Backend (API):** Proporciona la lógica de negocio y los servicios de datos necesarios para el funcionamiento de la aplicación.

2. Componentes Principales de la Arquitectura en detalle

Frontend: Next.js

El frontend está dividido en dos interfaces principales:

- Clientes: Desarrollado con Next.js, la interfaz para los usuarios permite realizar compras, gestionar carritos y pagar mediante Mercado Pago. Está optimizada para una experiencia móvil-first, con renderizado tanto del lado del cliente como del servidor, lo que mejora el rendimiento.

- Administradores: También construido en Next.js, este frontend está destinado a los administradores, quienes gestionan productos, pedidos y promociones. Los administradores acceden a este frontend mediante credenciales otorgadas manualmente. Este está adaptado a móvil pero se piensa como una aplicación de escritorio.

Backend: FastAPI (Python)

El backend se desarrolla con FastAPI, que maneja las solicitudes HTTP y gestiona las interacciones con la base de datos y otros servicios. Está diseñado para ser rápido y eficiente, permitiendo la escalabilidad y el manejo asíncrono de tareas, como el procesamiento de pagos o la gestión de productos. El backend se despliega en Google Cloud Run para asegurar un despliegue escalable y gestionar automáticamente los recursos según la demanda.

Infraestructura en la Nube:

- Google Cloud Run: El backend (FastAPI) será desplegado en Google Cloud Run, lo que permite escalar automáticamente según la demanda, sin necesidad de administrar servidores directamente.

- GCP Firestore: Base de datos NoSQL utilizada para almacenar los datos de la aplicación, como productos y pedidos, entre otros.

- Google Cloud Storage (Buckets): Se utilizan para almacenar archivos estáticos, como imágenes de productos.

- Google Cloud Secret Manager: Se utiliza para gestionar credenciales y secretos, como claves API o credenciales de Mercado Pago, de forma segura.
- GCP Pub/Sub: Para mensajería interna para desacoplar productores y consumidores y propagar eventos.
- Amazon SES: Para emails transaccionales (envío de resúmenes y confirmaciones).

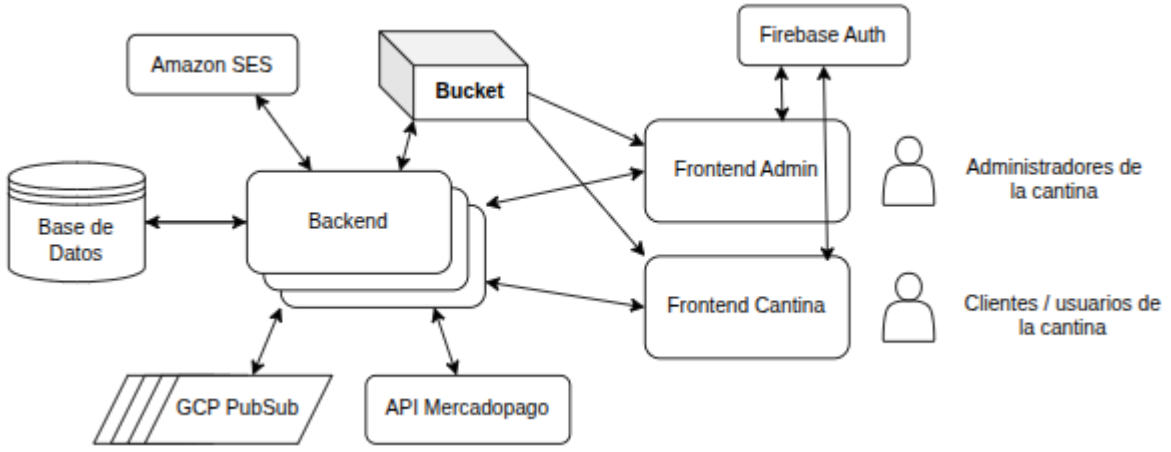
Servicios de Terceros

- Mercado Pago: Se integra con la aplicación para gestionar los pagos.
- Firebase Authentication: Proporciona autenticación y autorización para la gestión de usuarios en ambas interfaces (cliente y administrador).

Integración y Despliegue Continuo: GitHub Actions y Docker Hub

- GitHub Actions: Automatiza el flujo de trabajo de integración continua y despliegue continuo (CI/CD), incluyendo la construcción de la imagen de la aplicación y despliegue de la aplicación tanto en Cloud Run como en Vercel.
- Docker Hub: Se utiliza para almacenar las imágenes Docker de la aplicación backend, asegurando portabilidad y facilitando el despliegue en GCP.

3. Diagrama de la Arquitectura



4. Modelo de Comunicación

Usuario → Frontend (Cantina o Admin): Los usuarios (clientes o administradores) interactúan con la aplicación desde su navegador web.

- Los clientes acceden al Frontend Cantina para realizar pedidos, ver menús y recibir notificaciones en tiempo real.
- Los administradores utilizan el Frontend Admin para gestionar productos, pedidos y estadísticas.

Frontend ↔ Firebase Auth: Ambos frontends se comunican directamente con Firebase Authentication para gestionar el acceso y la identidad de los usuarios.

- Frontend Admin utiliza un proyecto de Firebase distinto al del frontend de clientes, con credenciales exclusivas para administradores.

Frontend → Backend (FastAPI en Cloud Run): Ambos frontends se comunican con el backend mediante una API RESTful implementada en FastAPI, utilizando peticiones HTTP y WebSockets para actualizaciones en tiempo real (sin recargar la página).

Backend → Firestore (Base de Datos): El backend consulta y actualiza la base de datos **Firestore** para manejar información de usuarios, productos, pedidos, ventas y estadísticas.

Backend → Bucket (Google Cloud Storage): El backend guarda y administra las imágenes (por ejemplo, fotos de productos o logos) en un bucket de Google Cloud Storage.

Frontend → Bucket (Google Cloud Storage): Los frontends consumen directamente esas imágenes desde el bucket, optimizando el rendimiento y reduciendo la carga sobre el backend.

Backend → Amazon SES: El backend utiliza Amazon Simple Email Service (SES) para enviar correos transaccionales a los usuarios (por ejemplo, confirmaciones de pedido, restablecimiento de contraseña o notificaciones administrativas).

Backend ↔ GCP Pub/Sub: Las distintas instancias del backend están **sincronizadas mediante GCP Pub/Sub**.

- Cuando ocurre un evento (por ejemplo, un nueva orden o cambio en el estado de la orden), el backend publica un mensaje en Pub/Sub.

- Todas las instancias del backend reciben ese mensaje, garantizando que cada una notifique correctamente a los usuarios conectados mediante WebSockets, incluso si están distribuidos entre diferentes instancias o IPs.

Backend ↔ API de Mercado Pago: El backend interactúa con la **API de Mercado Pago** para **crear enlaces de pago** asociados a cada pedido realizado por el usuario.

- Estos links de pago son devueltos al frontend para que el cliente pueda completar la transacción desde la interfaz web.

- Una vez realizado el pago, Mercado Pago envía una notificación al webhook configurado en el backend.

- El webhook procesa la confirmación de la transacción, valida el estado del pago y actualiza la información correspondiente en la base de datos (Se crea una orden y se borra la preOrden).

5. Consideraciones de Escalabilidad y Seguridad

- Escalabilidad: La arquitectura está diseñada para escalar automáticamente. Google Cloud Run ajusta los recursos según el tráfico de la aplicación, mientras que

Firestore es una base de datos altamente escalable que maneja cargas de trabajo intensivas.

- Seguridad: Los secretos y credenciales están protegidos usando Google Cloud Secret Manager, y Firebase Authentication asegura la gestión de usuarios con métodos de autenticación seguros. Además, los pagos son realizados únicamente a través de mercadopago, que asegura la seguridad de los mismos

3 Implementación

La implementación del proyecto se abordará de manera desestructurada, siguiendo una estrategia iterativa e incremental, guiada por la implementación de las historias de usuario.

Despliegue inicial

En primer lugar, se desplegará una versión mínima de la aplicación, que incluirá:

- Frontend básico: Se desplegarán ambos repositorios de la interfaz de usuario en Vercel (recordando que hay un frontend para clientes y otro para administradores de la cantina).

- Backend inicial: Se creará un pipeline de despliegue para la API del backend en Cloud Run, permitiendo un despliegue continuo. La primera versión incluirá un método HTTP GET en la ruta base (/) que devolverá información básica sobre la aplicación.

- Base de datos: Se configurará y desplegará la base de datos en Firestore, asegurando que esté disponible para el backend.

Desarrollo basado en historias de usuario

Una vez que la base del sistema esté operativa, se avanzará en la implementación de las historias de usuario. Cada historia se desarrollará siguiendo estos pasos:

1. Desarrollo o modificación de la interfaz en el frontend correspondiente, asegurando que el usuario pueda interactuar correctamente. Esto puede implicar la implementación de nuevos componentes, la modificación de un contexto, o ajustes en el layout de la aplicación en Next.js.

2. Implementación de endpoints en la API para obtener o enviar datos, asegurando la correcta manipulación de la información en el backend y la base de datos (crear, leer, actualizar o eliminar según sea necesario).

4 Pruebas

Para el proyecto Cantina UCC, se implementó una estrategia de pruebas centrada en la validación de la experiencia del usuario final, asegurando que cada funcionalidad cumpla con los requisitos y expectativas definidos. El enfoque principal fue la ejecución de **Pruebas**

Funcionales, un proceso diseñado para simular escenarios de uso reales y validar los flujos de trabajo críticos desde la perspectiva tanto de los clientes como de los administradores.

Con el fin de asegurar que el sistema satisficiera estas necesidades, el objetivo fue validar el cumplimiento de los requisitos funcionales descritos en las historias de usuario. Este enfoque permitió verificar de manera integral la usabilidad de las interfaces, el correcto funcionamiento del sistema y la comunicación entre el frontend (Next.js), el backend (FastAPI) y servicios esenciales como Firestore, Mercado Pago y Amazon SES.

A continuación, se detallan los casos de prueba diseñados para validar cada historia de usuario.

Casos de Prueba Funcionales:

Rol: Usuario Invitado

Historia de Usuario	Pasos a Ejecutar	Resultado Esperado
Quiero explorar el catálogo de productos.	1. Acceder a la página principal. 2. Observar la lista de productos. 3. Utilizar la barra de búsqueda para encontrar un producto. 4. Filtrar productos por categoría.	El catálogo se muestra correctamente con imágenes, nombres, descripciones y precios. La búsqueda y el filtro funcionan como se espera.
Quiero poder agregar productos al carrito.	1. Seleccionar un producto del catálogo y hacer clic en "Agregar". 2. Ir al carrito. 3. Aumentar la cantidad de un producto. 4. Eliminar un producto del carrito.	Los productos se añaden al carrito. El total y las cantidades se actualizan correctamente. Los productos se pueden eliminar.

Rol: Usuario Registrado

Historia de Usuario	Pasos a Ejecutar	Resultado Esperado
Quiero poder iniciar sesión utilizando mi correo electrónico o cuenta de Google.	1. Ir a la página de "Iniciar Sesión". 2.a Elegir “continuar con Google” e ingresar una cuenta de google. FIN 2.b Elegir “continuar con Email” 3.a Si ya tenes cuenta, ingresar correo y contraseña válida y hacer click en “iniciar sesión”. FIN 3.b Si no tenes cuenta elegir “crear cuenta”. 4. Usar un correo electrónico válido y hacer click en “enviar código de verificación”	El usuario puede acceder a una cuenta continuando con google o continuando con mail personal distinto a google, si nunca ha iniciado sesión, deberá crear una cuenta y luego podrá ingresar con mail y contraseña

	5. Ir al email ingresado y copiar el código, ingresarlo en el campo que lo pide y hacer click en “verificar código” 6. Luego ingresa una contraseña 2 veces y hacer click en “crear cuenta”. Con la cuenta creada ir al punto 3a	
Quiero poder realizar una compra y poner un horario para retirarlo.	1. Agregar productos al carrito, desde el catálogo o sumando cantidades en el carrito 2. Seleccionar un horario de retiro disponible. 3 Hacer click en “confirmar Carrito” 5. Hacer clic en "Pagar con Mercado Pago" y completar la transacción.	El pago se procesa exitosamente. La orden se genera en el sistema con el horario de retiro correcto.
Quiero recibir una confirmación por correo al finalizar una compra. con las instrucciones para retirar el pedido y la información de la transacción.	1. Realizar una compra exitosa (seguir los pasos del caso anterior). 2. Revisar la bandeja de entrada del correo asociado a la cuenta.	Se recibe un correo electrónico con el resumen del pedido, número de orden y horario de retiro.
Quiero poder comprar planes de comida por varios días. Para recibir descuentos	1. Ir a la sección "Planes de Comida". 2. Seleccionar un plan y la cantidad de días/menús. 3. Proceder al pago a través de Mercado Pago.	El pago se completa y el plan de comidas se acredita correctamente en la cuenta del usuario. Se realizan los descuentos correspondientes.
Quiero poder aplicar un plan de comida en una compra.	1. Asegurarse de tener un plan de comida activo (comprado previamente y no vencido) 2. Seleccionar en la pagina principal la opcion “comprar con planes de comida” 3. Seleccionar las comidas disponibles para el plan de comida 4. Proceder al checkout. (Elegir un horario y confirmar uso del plan)	El precio del producto se descuenta a cero. El sistema descuenta un uso del plan de comida activo. Se Puede ver una nueva orden con la comida seleccionada y el horario de retiro seleccionado
Quiero ver el historial de mis compras anteriores. para poder ver la información de todos mis pedidos y	1. Iniciar sesión. 2. Navegar a la sección "Mis Órdenes". 3. Revisar el listado de compras. 4. Navegar a "Mis Planes" para ver los planes adquiridos.	El historial muestra un listado correcto de todas las compras y planes, con sus detalles (fecha, total, estado).

mis planes de comida.		

Rol: Administrador de la Cantina

Historia de Usuario	Pasos a Ejecutar	Resultado Esperado
Quiero poder agregar, editar o eliminar productos.	1. Iniciar sesión como administrador. 2. Ir a "Productos" y hacer clic en "Agregar Producto". 3. Llenar el formulario y guardar. 4. Buscar el nuevo producto y hacer clic en "Editar". 5. Modificar el precio y guardar. 6. Eliminar el producto.	El producto se crea, actualiza y elimina correctamente. Los cambios se reflejan en el catálogo para clientes.
Quiero poder visualizar y administrar los pedidos de los usuarios.	1. Iniciar sesión como administrador. 2. Ir a la sección "Órdenes". 3. Filtrar órdenes por fecha y estado. 4. Cambiar el estado de una orden de "Pendiente" a "procesando" y luego a "Listo para retirar".	Las órdenes se muestran correctamente. Los filtros funcionan. El estado de la orden se actualiza y el cambio es visible para el cliente.
Quiero poder obtener un informe diario y mensual de las compras.	1. Iniciar sesión como administrador. 2. Ir a "Resúmenes". 3. Seleccionar la vista "Diaria" y verificar los montos. 4. Cambiar a la vista "Mensual" y verificar el total consolidado. 5. Realizar una compra en la cantina como un cliente. 6. Realizar la compra de un plan de comida como un cliente 7 Las compras se ven reflejada en el resumen diario y mensual	El sistema muestra los montos totales de ventas correctos para los periodos seleccionados, desglosando los productos vendidos.
Quiero poder imprimir los tickets de los pedidos.	1. En el listado de "Órdenes", seleccionar una orden. 2. Hacer clic en el botón "Imprimir Ticket".	Se abre el diálogo de impresión del navegador con un formato de ticket claro que contiene los detalles del pedido.
Quiero tener una pantalla con las órdenes para la cocina.	1. Iniciar sesión como administrador. 2. Navegar a la pantalla "Cocina"	La pantalla muestra las órdenes activas en un formato claro y fácil de leer para el personal de cocina, y se actualiza en tiempo real cuando

		desde el panel de “ordenes” enviamos una a la cocina o la retiramos de aca.
Quiero poder crear y gestionar los planes de comida	1. Navegar a la página "Planes de comida". 2. Hacer clic en una opción para "Crear Plan". 3. Llenar el formulario con los detalles del plan (ej. nombre, descripción, precio base, descuentos, fecha de vigencia). 4. Guardar el nuevo plan. 5. Buscar el plan recién creado en la lista y seleccionar "Editar" 6. Modificar todos los campos posibles y guardar los cambios.	El plan se crea exitosamente y se lista en la página de administración . El plan nuevo (y sus modificaciones) se refleja en la página "Planes de comida" del cliente.

Aclaración sobre el Entorno de Pruebas

Un aspecto fundamental de la estrategia fue la decisión de realizar las validaciones funcionales directamente sobre el entorno de producción. Es crucial aclarar este punto:

1. **Contexto del Proyecto:** El desarrollo fue llevado a cabo por un **único programador** como parte de un proyecto académico. No se trataba de un equipo de desarrollo gestionando un sistema comercial.
2. **Estado de la Aplicación:** Si bien la infraestructura estaba desplegada en un entorno "en vivo", la aplicación **aún no se encontraba en una fase de producción real**. No había sido lanzada a la comunidad universitaria, no tenía usuarios activos (más allá del desarrollador) y las únicas transacciones monetarias eran pruebas controladas con montos mínimos.

Bajo estas circunstancias, se tomó una decisión pragmática: mantener dos entornos idénticos (producción y *staging*) habría representado una **complejidad y un costo innecesarios** para un proyecto de esta escala y en esta etapa. El objetivo era agilizar la validación de funcionalidades en una configuración idéntica a la final, sin el riesgo de impactar a una base de usuarios que, en ese momento, era inexistente.

Reconocemos que esta práctica **no es el estándar en un entorno profesional** para sistemas ya operativos. Se subraya que, para el mantenimiento futuro y la implementación de nuevas funcionalidades una vez la aplicación esté en uso, es **imprescindible configurar un entorno de preproducción (o *staging*)**. Este entorno replicaría la infraestructura de producción y permitiría probar cualquier cambio de manera aislada antes de su lanzamiento definitivo, garantizando la estabilidad del servicio.

Otras aclaraciones

No se implementaron suites de pruebas unitarias o de integración automatizadas. La validación de la correcta interacción entre el frontend (Next.js), el backend (FastAPI) y los

servicios de terceros (Firestore, Mercado Pago) se abordó a través de las pruebas funcionales de los flujos completos de usuario.

IMPACTO ECONÓMICO

Para el impacto económico se estimaron los gastos operativos asociados con la infraestructura de nube (Google Cloud Platform - GCP, Vercel y Amazon SES) y el procesamiento de pagos (Mercado Pago) bajo tres escenarios de uso: 100, 1,000 y 5,000 compras diarias.

1. Metodología y Supuestos Clave

A continuación se detallan los supuestos fundamentales para la estimación de costos.

Escenarios Definidos:

Se evaluarán tres niveles operativos:

- Escenario 1 (Baja Escala): 100 compras/día (~3,000 compras/mes).
- Escenario 2 (Media Escala): 1,000 compras/día (~30,000 compras/mes).
- Escenario 3 (Alta Escala): 5,000 compras/día (~150,000 compras/mes).

Pila de Infraestructura:

- Backend y Base de Datos (GCP): Servicios en us-central1 (Iowa).
    - Cloud Firestore
    - Cloud Run.
    - Cloud Storage.
- Frontend (Vercel): Plataforma Vercel para Next.js.
- Amazon SES

Estimación de uso por compra

Para realizar una estimación de costos precisa y relevante, se establecieron una serie de supuestos sobre el consumo de recursos por cada "compra" o "sesión de usuario" exitosa. Estas cifras representan un promedio agregado de las interacciones necesarias para completar un pedido, desde la navegación hasta la confirmación, y tienen en cuenta las particularidades de la arquitectura implementada.

<u>Consumo GCP por Compra:</u>

Los siguientes valores son estimaciones promedio por cada compra exitosa en la aplicación, cubriendo tanto las operaciones del cliente como las de los administradores:

Lecturas Firestore: 85

Una única compra implica múltiples operaciones de lectura en la base de datos a lo largo del recorrido del usuario, el proceso del backend, y las interacciones post-compra. Esto incluye:

- Fase Pre-compra y durante la compra (aproximadamente 75+ lecturas):
    - Lectura del catálogo de productos (inicial y posibles filtros/búsquedas, que pueden ser varias lecturas). (50 o más productos)
    - Lectura de productos en la pantalla del carrito de compras de compras.
    - Lectura de la información del usuario para el checkout.
    - Lecturas asociadas a la lógica de negocio para validar o actualizar planes de comida (ej., verificar existencia de plan).
- Fase Post-compra (aproximadamente 8-10 lecturas):
    - Lectura de para la confirmación enviada por email.
    - Lectura del historial de pedidos del usuario (Mis Órdenes), donde la nueva compra aparecerá. Se asume que el usuario revisará su historial después de una o varias compras.
    - Lecturas para los administradores de la cantina al visualizar las nuevas órdenes (Órdenes y Cocina), lo cual puede implicar varias lecturas para filtrar, ordenar o ver detalles.
    - Lectura del estado actualizado del FoodPlan o OrdersForPlans si se utilizó un plan de comida, para reflejar el consumo.

Este número (85) es una estimación agregada que intenta cubrir el flujo completo de una compra exitosa y las operaciones auxiliares necesarias, tanto inmediatas como posteriores, tanto del cliente como de la administración, para considerar todo el ciclo de vida de la información de esa compra.

Escrituras Firestore: 10

Las operaciones de escritura son fundamentales para registrar el estado de la compra y actualizar los datos. Esto incluye:

- Creación de la PreOrden con sus detalles.
- Varias escrituras post-pago, que son cruciales:
    - Creación de la Order definitiva (con sus ítems, estado, etc.).
    - Actualización del DayOrderCounter.
    - Actualización o creación de DailySummaries y MonthlySummaries (consolidados de ventas).
    - Actualización del OrdersForPlans o decremento de unidades de un FoodPlan si se utiliza.

    - Eliminación de la PreOrden tras la confirmación del pago.

Cada una de estas acciones puede implicar una o más escrituras, justificando el promedio de 10 por compra.

Solicitudes Cloud Run: 15

Cada interacción del frontend con el backend se traduce en una solicitud HTTP a Cloud Run. Una compra promedio involucra:

- Una solicitud para cargar el catálogo, detalles de productos del carrito.
- La solicitud para iniciar el proceso de pago con Mercado Pago (generar la URL de pago).
- Una solicitud crucial: el webhook de Mercado Pago que notifica al backend sobre el estado del pago.
- Solicitudes para verificar el estado de la orden o acceder al historial post-compra.
- Solicitudes de la interfaz de administración para ver órdenes recientes o resúmenes diarios.

El número 15 es un promedio que abarca la ruta completa de un usuario desde la navegación inicial hasta la confirmación del pedido, incluyendo las interacciones de los servicios (como el webhook) y las consultas subsiguientes relevantes.

Egreso Cloud Storage: 10MB (hacia Sudamérica)

El Egreso (o transferencia de datos saliente) es el cargo que se aplica por el volumen de datos (en GB) que sale de la red del proveedor de la nube desde la región de origen hacia un destino externo, como: Internet (que incluye a tus usuarios en Sudamérica) u otra región de la nube.

En nuestro caso, el "egreso a Sudamérica" es el costo directo por cada GB que los usuarios sudamericanos descargan o acceden desde el almacenamiento.

Una sesión de compra no solo carga una imagen, sino que el usuario navega por el catálogo de productos. Los 10 MB representan un catálogo con aproximadamente unas 35 imágenes.

Operaciones de Clase B: 40

Las Operaciones de Clase B son cargos que se aplican por ciertas solicitudes realizadas a tu almacenamiento en la nube que generalmente implican la lectura del estado existente de los objetos o metadatos, pero no modifican el estado de los datos.

Son consideradas operaciones de "baja frecuencia" o de menor impacto en los recursos del sistema en comparación con las de Clase A (Operaciones de mutación de estado o de alto consumo. (Ej.: Subir objetos Insert/Update)).

Cuando un usuario en Sudamérica descarga un archivo (por ejemplo, de 5 MB) de nuestro bucket:

1. Se registra una operación de Clase B (la solicitud Get para obtener el archivo). Clase B cobra por el acto de solicitar el dato (la llamada API, independientemente de dónde vaya el dato).

2. Se registra una transferencia de salida (Egress) de 5 MB dirigida a Sudamérica. El egreso cobra por el volumen del dato transferido fuera de la región (los GB descargados).

Las 40 operaciones por compra se justifican para un catálogo de 35 productos más la vista de imágenes en resúmenes de orden por parte del admin y del cliente

Resumen consumos GCP:

- Lecturas Firestore: 85.
- Escrituras Firestore: 20.
- Solicitudes Cloud Run: 15.
- Egreso Cloud Storage: 10MB (hacia Sudamérica).
- Operaciones Cloud Storage (Clase B): 40.

Uso de Vercel (frontend):

El frontend de Cantina UCC está desplegado en Vercel (Next.js), una plataforma que utiliza un modelo de costos basado en planes fijos que incluyen grandes paquetes de servicios clave, como la transferencia de datos y las solicitudes.

La única métrica que se espera que tenga un impacto financiero significativo en los escenarios definidos es el Fast Data Transfer (FDT), que mide el volumen de datos que la aplicación entrega desde la red global de Vercel hacia los usuarios finales.

No se espera que otras métricas generen un costo significativo dentro del rango normal de operación, dado que la aplicación está diseñada para manejar compras internas de una universidad y no grandes picos de tráfico viral. Sin embargo, estas métricas deben ser monitoreadas en la puesta en producción de la aplicación:

Métrica	**Límite del Plan Pro**	**Riesgo de Costo y Escenario de Disparo**
Edge Requests (Solicitudes al Edge)	10 millones por mes	Riesgo bajo. El escenario de mayor uso (E3: 150k compras/mes) genera solo ~2.25M de solicitudes. El límite de 10 M es muy alto para la carga de trabajo esperada, y el excedente es marginal ($2.00 por millón [9]).

Fast Origin Transfer (FOT)	10 GB gratuito (luego pago por uso)	Riesgo medio. Si la configuración del caché (ISR/Headers) falla, se podría generar una alta transferencia de datos desde el backend de GCP a Vercel. Un FOT elevado indica un fallo de eficiencia que debe corregirse, más que un problema de escala.
Image Optimization - Transformations	300 KB (plan gratuito)	Riesgo bajo. Solo se dispara cuando se suben y transforman nuevas imágenes. Dado que el menú de la cantina no cambia radicalmente todos los días, se espera que este costo sea despreciable.
Function Invocations / CPU Duration	Gran volumen incluido	Riesgo muy bajo. La lógica pesada de negocio y pagos reside en el backend de GCP. El uso de funciones serverless de Vercel para el frontend debería ser mínimo.

Firebase Authentication

Se asume < 50,000 MAU (usuarios activos por mes) en todos los escenarios, manteniéndose dentro del nivel gratuito. Superar este límite activaría costos adicionales.

Comisiones de Mercado Pago

Se analizan las tasas porcentuales para Checkout/Link de Pago en Argentina.

Consumo amazon SES por Compra

3 emails (confirmación de compra, estado de orden lista, estado de orden entregada)

2. Análisis de Costos Estimados por Componente

Costos de GCP (us-central1)

El cálculo de costos se deriva directamente de la identificación del volumen de sobreuso para cada servicio, una vez agotada la cuota gratuita.

Tabla resumen de los umbrales clave de la capa gratuita (Free Tier)

Componente GCP	**Límite Diario (FD)**	**Límite Mensual (30 días)**
Cloud Run Solicitudes	N/A	2,000,000
Firestore Lecturas	50,000	1,500,000
Firestore Escrituras	20,000	600,000
Storage Ops Clase B	N/A	50,000

Egresos de Red (Internet)	N/A	200 GiB

La gestión de la Capa Gratuita de Firestore requiere una mención específica. Sus límites están definidos diariamente: 50,000 lecturas y 20,000 escrituras. Asumiendo un consumo perfectamente uniforme a lo largo del mes, el límite efectivo de 30 días es de 1,500,000 lecturas y 600,000 escrituras. Si el patrón de uso no fuera uniforme (por ejemplo, picos de tráfico), cualquier consumo que exceda la cuota diaria sería facturable inmediatamente, incluso si el total mensual se mantiene cerca del límite agregado.

Las tarifas base aplicadas para el cálculo del excedente son las siguientes, basadas en los precios predeterminados de GCP:

Componente GCP	**Tarifa Unitaria (USD)**	**Unidad de Medida**
Cloud Run Solicitudes	$0.40	Por 1,000,000 de Peticiones
Firestore Lecturas	$0.03	Por 100,000 Lecturas
Firestore Escrituras	$0.09	Por 100,000 Escrituras
Storage Ops Clase B	$0.0004	Por 1,000 Operaciones
Egresos de Red (Internet)	$0.085	Por GiB (Gigibyte)

Se procede a calcular el excedente de uso para cada componente y escenario, aplicando las tarifas definidas:

Escenario de Consumo Bajo (3k Compras/Mes)

El volumen total de consumo es marginal:

- Cloud Run 45k
- Firestore Lecturas 255k
- Firestore Escrituras 60k
- Storage Ops B 60k
- Egreso 29.3 GiB

Componente	**Consumo Total**	**Límite Gratuito**	**Excedente**	**Costo Unitario**	**Costo Mensual**
Cloud Run Rqs	45k	2.0 M	0	$0.40/M	$0.00

Firestore Lecturas	255k	1.5 M	0	$30.00/M	$0.00
Firestore Escrituras	60k	0.6 M	0	$90.00/M	$0.00
Storage Ops B	60k	50k	10k	$0.40/M	$0.0004
Egreso de Red	29.3 GiB	200 GiB	0	$0.085/GiB	$0.00

El costo total facturado es de $0.00 (después del redondeo del excedente mínimo en Storage Ops B). Este nivel de tráfico es ideal para la fase de desarrollo, pruebas de concepto, o aplicaciones de bajo volumen, ya que opera casi en su totalidad dentro de la capa gratuita ofrecida por GCP.

Escenario de Consumo Medio (30k Compras/Mes)

El consumo total se incrementa a:

- Cloud Run 450k
- Firestore Lecturas 2.55 M
- Firestore Escrituras 0.6 M
- Storage Ops B 0.6 M
- Egreso 293.0 GiB

Cálculos Detallados del Excedente Facturable:

1. Cloud Run Solicitudes: 450k solicitudes se mantienen muy por debajo del límite de 2.0 M. **Costo: $0.00.**
2. Firestore Lecturas:
    - Consumo Total: 2.55 M. Límite Gratuito: 1.5 M.
    - Excedente Billable: $2,550,000 - 1,500,000 = 1,050,000 Lecturas.
    - Costo: (1,050,000/100,000)×$0.03=10.5×$0.03=**$0.32.**
3. Firestore Escrituras:
    - Consumo Total: 0.6 M (600k). Límite Gratuito: 0.6 M.
    - Excedente Billable: 0. **Costo: $0.00.**
4. Storage Ops Clase B:

- Consumo Total: 0.6 M (600k). Límite Gratuito: 50k (0.05 M).
- Excedente Billable: $600,000 - 50,000 = 550,000 Operaciones.
- Costo: (550,000/1,000,000)×$0.40=0.55×$0.40=**$0.22.**

5. Egreso de Red (GiB):
    - Consumo Total: 293.0 GiB. Límite Gratuito: 200 GiB.
    - Excedente Billable: 293.0 GiB−200.0 GiB=93.0 GiB.
    - Costo: 93.0 GiB×$0.085/GiB=**$7.91.**

El costo total facturado es de $8.45. En este nivel, la aplicación ha superado el umbral de la Capa Gratuita y el costo del Egreso de Red ya se establece como el impulsor de costos dominante.

Escenario de Consumo Alto (150k Compras/Mes)

El consumo total escala significativamente

- Cloud Run 2.25 M
- Firestore Lecturas 12.75 M
- Firestore Escrituras 3.0 M
- Storage Ops B 3.0 M
- Egreso 1,464.8 GiB.

Cálculos Detallados del Excedente Facturable:

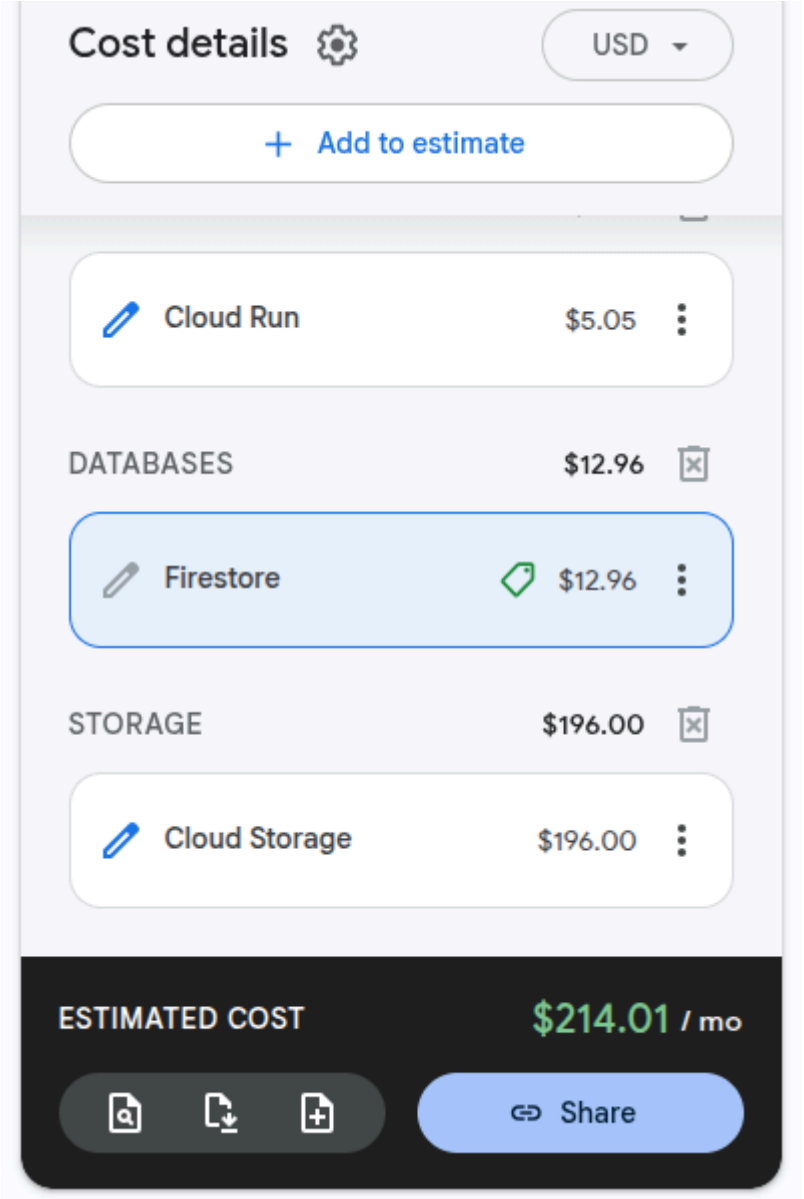
1. Cloud Run Solicitudes:
    - Excedente Billable: 2.25 M−2.0 M=0.25 M Solicitudes.
    - Costo: (0.25 M/1 M)×$0.40=$**0.10.**
2. Firestore Lecturas:
    - Excedente Billable: 12.75 M−1.5 M=11.25 M Lecturas.
    - Costo: (11.25 M/100,000)×$0.03=112.5×$0.03=**$3.38.**
3. Firestore Escrituras:
    - Excedente Billable: 3.0 M−0.6 M=2.4 M Escrituras.
    - Costo: (2.4 M/100,000)×$0.09=24.0×$0.09=$2.16.
    - Costo Total Firestore: $3.38+$2.16=**$5.54.**
4. Storage Ops Clase B:
    - Excedente Billable: 3.0 M−0.05 M=2.95 M Operaciones.
    - Costo: (2.95 M/1 M)×$0.40=2.95×$0.40=**$1.18.**

5. Egreso de Red (GiB):
    - Excedente Billable: 1,464.8 GiB−200.0 GiB=1,264.8 GiB.
    - La tarifa de $0.085/GiB aplica para el tramo de 200 GiB a 10,240 GiB.
    - Costo: 1,264.8 GiB×$0.085/GiB=**$107.51**.

El costo total asciende a $114.33, confirmando la predominancia del Egreso de Red, que consume casi la totalidad del presupuesto operativo.

Corrección del 3er escenario

Sin embargo, al utilizar la calculadora de precios de Google Cloud Platform con los mismos parámetros de consumo, el costo total estimado para el tercer escenario asciende a **USD 214**. **Link a los resultados del cálculo**



Por lo que vamos a tomar este valor como valor final para el 5to escenario por que suponemos que esta teniendo mas cosas en cuenta que se nos pueden haber pasado de largo

Costos de Vercel

Teniendo en cuenta que la única métrica analizada es el FDT, los costos en los distintos escenarios son los siguientes:

Métrica	**Escenario 1 (100 compras/día)**	**Escenario 2 (1,000 compras/día)**	**Escenario 3 (5,000 compras/día)**
FDT Consumo Mensual	30 GB	300 GB	1,500 GB
Límite Plan Hobby	100 GB incluido	Superado (Requiere Plan Pro)	Superado (Requiere Plan Pro)
Límite Plan Pro	1 TB (1,024 GB) incluido	1 TB (1,024 GB) incluido	Superado (476 GB de excedente)
Costo FDT Adicional	$0.00	$0.00	$104.72
Total Vercel	**$0.00**	**$20.00 (Plan Base)**	**$124.72**

Costos de Mercadopago

El costo es un porcentaje variable sobre las ventas, dependiente del plazo de acreditación elegido.

Estructura de Tasas (Incluyendo 21% IVA): Las tasas efectivas totales para Checkout/Link de Pago en Argentina son :

- Inmediata: 7.61%
- 10 Días: 5.31%
- 18 Días: 4.10%
- 35 Días: 1.80%

Tabla de tasas de comisión de Mercado Pago

Plazo de Disponibilidad	**Tasa Base (%)**	**IVA (21%)**	**Tasa Total (%)**
Al instante	6.29%	1.32%	**7.61%**

10 días	4.39%	0.92%	**5.31%**
18 días	3.39%	0.71%	**4.10%**
35 días	1.49%	0.31%	**1.80%**

Este costo escala linealmente con la facturación y será el componente dominante del gasto operativo a medida que la aplicación crezca.

Costos de Amazon SES

Emails por compra

Cada compra genera **3 correos**.

- Escenario 1: 100 compras/día → 300 emails/día → ~9,000 emails/mes.
- Escenario 2: 1,000 compras/día → 3,000 emails/día → ~90,000 emails/mes.
- Escenario 3: 5,000 compras/día → 15,000 emails/día → ~450,000 emails/mes.

Costo = (emails/mes ÷ 1,000) × $0.10 USD

1. **Escenario 1 – Baja escala**
   9,000 emails/mes → (9,000 ÷ 1,000) × $0.10 = **$0.90 USD/mes**
2. **Escenario 2 – Media escala**
   90,000 emails/mes → (90,000 ÷ 1,000) × $0.10 = **$9 USD/mes**
3. **Escenario 3 – Alta escala**
   450,000 emails/mes → (450,000 ÷ 1,000) × $0.10 = **$45 USD/mes**

4. Análisis Costo-Beneficio Conciso

- **Costos de Infraestructura (GCP + Vercel + amazon SES):** El riesgo financiero es bajo. Los costos varían desde ~$**0.90**/mes (100 compras/día) hasta ~$**383**/mes (5,000 compras/día). La infraestructura es asequible y escalable. La principal decisión es la transición al plan Pro de Vercel (al superar aprox las 250 compras/día).
- **Costos de Procesamiento de Pagos (Mercado Pago):** Es el factor de costo principal y variable. Las comisiones (entre 1.80% y 7.61% según el plazo de entrega en mayo 2025) impactan directamente la rentabilidad. La elección del plazo de acreditación es una decisión estratégica clave (margen vs. flujo de caja).
- **Viabilidad:** La viabilidad económica depende del volumen de ventas y del margen por transacción para cubrir las comisiones de Mercado Pago. Sin embargo la cantina

ya trabaja con cobros a través de Mercadopago QR que aplica las mismas tasas de cobro, por lo que no que ya deberían estar adaptados a las comisiones que aplica esta plataforma. Por otro lado, la infraestructura no es una barrera significativa en estos niveles de escala.

- **Gasto en relación a las Ventas (En el escenario de mayor gasto 5,000 compras/día):** Asumiendo un promedio de $2,000 ARS por compra, el costo total de infraestructura (~$383 USD/mes) representa aproximadamente sólo el **0.195%** del volumen de ventas mensual ($2.000 x 5.000 x 30 = $300.000.000 ARS, o ~$200.000 USD con dolar a $1500). Por lo que sea cual sea el escenario, el aumento del consumo de recursos y de los costos en los servidores se justifica automáticamente por el aumento de las ventas que disparan estos costos. Haciendo el mismo cálculo suponiendo que los precios se nos dispararan a $1000 USD y manteniendo el nivel de ventas (suponiendo que subestimamos los costos de la infraestructura), ese monto (los 1.000 USD) representarían un **0.5%** del monto total de ventas. Una cifra más que aceptable en comparación con las comisiones que cobra mercadopago

- **Potencial de Optimización de GCP:** Si se configura adecuadamente el caché de las imágenes del catálogo de productos en Vercel, asegurando que solo se soliciten las imágenes al bucket de Cloud Storage una vez al año (o cuando la imagen se actualiza, cambiando su nombre), el componente de Egreso de Red de GCP ($107.51) se podría eliminar, reduciendo el costo total de GCP a ~$18/mes en el Escenario 3.

- Volviendo a recalcar, si la configuración del servidor es adecuada (no genera gastos innecesarios), cualquier gasto infraestructural se verá justificado por el incremento correspondiente en las ventas; no obstante, dicho gasto puede monitorearse y optimizarse para maximizar la eficiencia del sistema.

5. Para tener en cuenta

- Optimizar Mercadopago: Analizar flujo de caja vs. margen para elegir el plazo de acreditación óptimo.

- Gestionar Costos en GCP: Usar Alertas de Presupuesto para monitorear (no limitar) el gasto. Considerar Cuotas para limitar proactivamente el uso de recursos si es necesario.

- Gestionar Costos en Vercel: Monitorear FDT (Fast Data Transfer) en el Dashboard de Usage. Optimizar imágenes/frontend para retrasar la necesidad del plan Pro. En Pro, usar Spend Management para alertas o pausas automáticas.

6. Resumen de Resultados y Conclusión Final

Resumen de Costos:

- Infraestructura (GCP + Vercel + SES):
    - 100 compras/día: ~$0.90 USD/mes.

    - 1,000 compras/día: ~$37 USD/mes.
    - 5,000 compras/día: ~$383 USD/mes.

- Procesamiento de Pagos (Mercado Pago): Costo variable entre 1.80% y 7.61% del valor de cada transacción.

El análisis detallado demuestra que *Cantina UCC* es un proyecto económicamente factible y altamente escalable desde el punto de vista operativo. La infraestructura tecnológica basada en Google Cloud Platform (GCP) y Vercel permite soportar desde 0 hasta 5.000 compras diarias con costos mensuales que oscilan entre $0.90 y $383 USD, dependiendo del volumen de uso. Esta estructura de costos progresiva y predecible ofrece una base sólida para el crecimiento sostenido del sistema, sin representar un riesgo financiero significativo en las etapas iniciales.

El principal componente variable en los costos operativos es la comisión de Mercado Pago, que depende directamente del volumen de ventas y del plazo de acreditación elegido. Este aspecto, sin embargo, puede ser optimizado estratégicamente según las necesidades de flujo de caja o márgenes de ganancia esperados. La existencia de múltiples opciones de acreditación con tasas diferenciadas permite a los administradores del sistema adaptar el modelo económico en función de la situación particular del negocio.

Además, la posibilidad de permanecer dentro de los niveles gratuitos en GCP hasta escalas considerables (por ejemplo, hasta 1.000 compras diarias) reduce significativamente la barrera de entrada, facilitando el lanzamiento y consolidación del servicio sin requerir una inversión inicial significativa en infraestructura.

A estos factores se suma que ciertos costos pueden ser compensados indirectamente a través de:

- Un posible aumento en las ventas, producto de una experiencia de compra más ágil, y la fidelización de clientes al incluir los planes de comida.

- Beneficios no tangibles que, aunque difíciles de cuantificar, generan un alto valor para la institución: reducción de filas, inclusión digital, conciencia ecológica, fortalecimiento del compromiso social y mejora del clima institucional. Estos elementos contribuyen a consolidar una imagen positiva del servicio, a justificar su implementación y a facilitar el apoyo institucional o incluso la financiación externa.

RSU

La Responsabilidad Social Universitaria (RSU) en la Universidad Católica de Córdoba (UCC) se entiende como la capacidad y compromiso institucional para responder a las necesidades de transformación de la sociedad en la que está inmersa, a través del ejercicio de sus funciones sustantivas: docencia, investigación, extensión y gestión interna.

En este marco, el proyecto de tesis "Cantina UCC" se alinea plenamente con los principios de la RSU al abordar una problemática concreta dentro de la comunidad universitaria: las largas filas y tiempos de espera en la cantina durante los horarios de mayor afluencia. A través del desarrollo de una aplicación web responsive, que permite realizar pedidos y pagos anticipados, se busca mejorar la calidad de vida de estudiantes, docentes y personal administrativo, optimizando su tiempo y su experiencia en el campus.

Además, el proyecto promueve la inclusión digital, ya que ofrece una plataforma intuitiva y accesible para todo tipo de usuarios, independientemente de su familiaridad con la tecnología.

En el plano ambiental, Cantina UCC también contribuye activamente a la sostenibilidad al eliminar la necesidad de emitir tickets físicos. De este modo, se reducen significativamente el consumo de papel, agua, energía y las emisiones de CO_2 asociadas a la producción de papel térmico. La aplicación no solo digitaliza el proceso de compra, sino que además promueve activamente la conciencia ambiental entre sus usuarios: con cada compra, informa sobre el ahorro de recursos generado, generando así un efecto educativo y de sensibilización sobre el impacto positivo que tiene adoptar hábitos más sostenibles en la vida diaria.

Por todas estas razones, el proyecto "Cantina UCC" representa una solución concreta y aplicada de Responsabilidad Social Universitaria, en tanto articula tecnología, servicio y valores institucionales en un mismo desarrollo. Si bien su alcance es limitado al entorno inmediato del campus, logra integrar de manera equilibrada dimensiones sociales, organizacionales, educativas y ambientales. No se trata solamente de una solución técnica a un problema cotidiano, sino de una propuesta que busca generar valor para la comunidad universitaria, contribuyendo a una mejor calidad de vida, a una cultura de solidaridad y a prácticas más sostenibles, en coherencia con el compromiso formativo y transformador que la UCC promueve desde su modelo institucional.

En las secciones dedicadas al impacto social y al impacto ambiental, se profundizará con mayor detalle en las implicancias concretas que este proyecto tiene en cada uno de estos ámbitos.

IMPACTO SOCIAL

Beneficio o Impacto Positivo General

Cantina UCC representa una mejora en la calidad de vida de la comunidad universitaria. Al optimizar el proceso de compra y pago de alimentos en la cantina, la aplicación reducirá los tiempos de espera, mejorará la experiencia del usuario y permitirá un uso más eficiente del tiempo, especialmente en contextos de recreos breves. Esto contribuirá a un entorno académico más saludable, ordenado y accesible, donde los estudiantes, docentes y personal administrativo puedan disfrutar de sus tiempos de descanso con mayor efectividad y tranquilidad.

Segmentos de la Población Beneficiados

El principal grupo beneficiado por esta solución serán los estudiantes universitarios, quienes muchas veces disponen de recreos muy cortos para almorzar. También se verán favorecidos los docentes y el personal administrativo, que podrán evitar largas filas y acceder a un servicio más eficiente. Asimismo, el equipo de administración de la cantina se beneficia al contar con herramientas que optimizarán la operación diaria, reducirán la carga de trabajo en horas pico y permitirán prever la demanda.

Inclusión y Reducción de Brechas

Cantina UCC promueve la inclusión digital al acercar herramientas tecnológicas a un servicio tradicional. Este enfoque permite reducir la brecha entre quienes están acostumbrados a sistemas digitales y quienes no, ya que la plataforma ha sido pensada para ser intuitiva y accesible para todo tipo de usuarios y teléfonos móviles, independientemente de su familiaridad con la tecnología. Además, al ofrecer opciones de compra anticipada y digital, se garantiza mayor equidad en el acceso al servicio, sin importar la disponibilidad horaria de cada estudiante.

IMPACTO MEDIOAMBIENTAL

El proyecto **Cantina UCC** contribuye activamente a la sostenibilidad medioambiental mediante distintas acciones enfocadas en reducir residuos, optimizar recursos y operar sobre una infraestructura más limpia:

Eliminación de tickets impresos

La digitalización total del sistema de compra elimina la necesidad de emitir tickets físicos. Esto representa un ahorro directo de recursos cada vez que un ticket no es impreso.

Ahorro por 1 ticket no impreso:

- Papel: 0.2 gramos - Un ticket promedio de 10–12 cm pesa entre 0.15 y 0.2 gramos, según el gramaje del papel térmico (48–55 g/m²).
- Madera cruda evitada: 0.4 a 0.48 gramos - Según diversas fuentes, para fabricar una tonelada de papel virgen se requieren aproximadamente entre 2.400 y 2.700 kilogramos de madera, lo que equivale a unos 17 árboles adultos. Dado que un ticket promedio pesa alrededor de 0,2 gramos, al evitar la impresión de un solo ticket se ahorran aproximadamente 0,4 a 0,48 gramos de madera cruda.
- Agua utilizada para producir el papel: 25 a 50 ml - La producción de papel consume entre 125 y 250 litros de agua por kilogramo de papel. Cálculo: 0.0002 kg de papel × 125–250 L/kg = 0.025–0.05 L = 25–50 ml
- Energía utilizada en producción e impresión: ~0.0018 kWh - La fabricación de papel consume entre 6 y 12 kWh por kilogramo. Cálculo: 0.0002 kg de papel × 9 kWh/kg (considerando producción y procesamiento) = 0.0018 kWh
- Emisiones de CO_2: ~2 gramos - La producción de papel genera entre 0.7 y 1.2 kg de CO_2 por kilogramo de papel. Cálculo: 0.0002 kg de papel × 10 kg CO_2/kg = 0.002 kg = 2 g

Ahorro acumulado por 1000 tickets no impresos:

- **Papel:** 200 gramos
- **Madera cruda evitada:** 400 a 480 gramos
- **Agua utilizada para producir el papel:** 25 a 50 Litros
- **Energía utilizada en producción e impresión:** 1,8 kWh
- **Emisiones de CO_2:** ~2 kg

Dato clave extra*:* El papel térmico usado en tickets no es reciclable y suele terminar como residuo no recuperable, aumentando el impacto ambiental a largo plazo.

¿Qué se puede hacer con 1 kWh?

Dispositivos personales:

- Cargar un celular hasta 75 veces
- Usar una notebook durante 20 a 25 horas
- Escuchar música con auriculares bluetooth por más de 300 horas

Cocina y hogar:

- Hervir 10 litros de agua en una pava eléctrica
- Tostar pan durante 1 hora continua
- Usar una cocina eléctrica de 1000 W durante 1 hora
- Lavar una carga de ropa en frío (lavarropas eficiente)

Electrodomésticos:

- Hacer funcionar una heladera eficiente por casi 1 día
- Encender una lámpara LED de 10W durante 100 horas
- Alimentar una TV LED de 100W por unas 10 horas

Transporte eléctrico:

- Recargar una bicicleta eléctrica completamente una vez
- Recorrer 6 a 8 km con un auto eléctrico (promedio de 15 kWh/100 km)

Concientización sobre la eliminación de tickets impresos

La app no solo digitaliza la experiencia de compra, sino que también informa a los usuarios sobre el impacto positivo de evitar los tickets físicos. Con cada compra realizada, te mostramos cuántos recursos estás ayudando a ahorrar (papel, agua, energía y emisiones) y te recordamos que cada pequeño gesto suma en el cuidado del planeta. Elegir no imprimir un ticket es elegir un futuro más sustentable.

Reducción del desperdicio alimentario

La funcionalidad de **Planes de Comida** y **Compra anticipada**, permite a los usuarios reservar sus comidas con antelación. Esto habilitará a la cantina a planificar mejor la cantidad de alimentos a preparar cada día, disminuyendo el excedente y el desperdicio de comida.

Infraestructura sustentable

Actualmente, la aplicación está alojada en **Google Cloud**, una plataforma que opera con **energía 100% renovable** en todos sus centros de datos. Esto significa que la operación del backend (incluyendo la base de datos y API) tiene una **huella de carbono prácticamente**

nula, alineándose con las prácticas más sostenibles del sector. **Anexo 5**

BENEFICIOS POST IMPLEMENTACIÓN

Una vez implementado el sistema Cantina UCC, se espera obtener una serie de beneficios tangibles e intangibles que impactarán positivamente tanto en la administración de la cantina como en la comunidad universitaria.

Beneficios Tangibles (Operativos y Financieros):

- **Aumento de la Eficiencia Operativa:** El beneficio más significativo será la optimización de la operatividad de la cantina, especialmente durante las horas pico. Al digitalizar y desacoplar el proceso de pago del proceso de retiro, se elimina el principal cuello de botella identificado: la congestión en la caja. Esto permitirá procesar un mayor volumen de pedidos en menos tiempo.
- **Reducción del Desperdicio Alimentario:** La implementación de pedidos anticipados y planes de comida proporcionará a la administración herramientas para prever la demanda diaria. Esto permitirá a la cocina planificar de manera más precisa la cantidad de alimentos a preparar , reduciendo el excedente y el desperdicio.
- **Automatización de Tareas Administrativas:** El sistema generará automáticamente resúmenes de compras diarias y mensuales. Esto elimina la necesidad de realizar cierres de caja y registros manuales, reduciendo la carga de trabajo administrativo y minimizando errores.
- **Recuperación de Ventas Perdidas:** Se espera un incremento en las ventas al recapturar a los clientes (estudiantes y docentes) que anteriormente decidían no comprar debido a las largas filas.
- **Fidelización de Clientes:** La gestión digital de "Planes de Comida" fomenta la compra anticipada con descuento , asegurando un flujo de ingresos recurrentes y fomentando la lealtad de los usuarios.

Beneficios Intangibles (Experiencia y Estrategia):

- **Mejora en la Calidad de Vida del Campus:** Se reducirán drásticamente los tiempos de espera , mejorando la experiencia general de estudiantes y docentes. Esto permite un uso más eficiente del tiempo de descanso , reduciendo la frustración y la desmotivación asociadas a las filas.
- **Mejora en la Toma de Decisiones:** Los administradores tendrán acceso a datos consolidados sobre ventas y productos. Esta información facilitará la toma de decisiones estratégicas sobre el catálogo de productos y la planificación de la demanda.
- **Modernización del Servicio:** La implementación de una plataforma digital alinea el servicio de la cantina con las expectativas modernas de los usuarios, promoviendo la inclusión digital y mejorando la imagen general del servicio.

CONCLUSIÓN

El desarrollo del proyecto ha representado una valiosa y exhaustiva instancia de aprendizaje integral, permitiéndome no solo abordar, sino también navegar con éxito el ciclo completo de vida de una aplicación web moderna, desde su concepción y diseño estratégico hasta su despliegue final en una infraestructura de nube. La experiencia me ha permitido adquirir y consolidar conocimientos prácticos fundamentales en la orquestación de un desarrollo complejo, abarcando de manera integrada tanto el frontend, con sus desafíos de usabilidad y experiencia de usuario, como el backend y la robusta infraestructura en la nube que lo soporta.

Durante el proceso, una de las reflexiones recurrentes fue la posibilidad de haber construido una aplicación móvil nativa. Si bien la familiaridad con tecnologías como React Native habría hecho esta alternativa aún más tentadora, el análisis profundo del contexto de uso me llevó a la conclusión de que la simplicidad y la accesibilidad de una app web son sus mayores fortalezas en este caso. La decisión de optar por un sistema fácilmente accesible mediante un código QR fue una elección estratégica para minimizar la barrera de entrada, evitando forzar a estudiantes o docentes a buscar, descargar e instalar una aplicación que podrían utilizar de forma esporádica.

Uno de los aprendizajes técnicos más significativos fue, sin duda, la implementación de una arquitectura desplegada en Google Cloud Platform. Esta fase del proyecto me brindó una experiencia práctica invaluable en la configuración y gestión de servicios clave como Cloud Run para el despliegue escalable de la API y Firestore como base de datos NoSQL en tiempo real. La elección de esta plataforma, motivada por su escalabilidad y su generosa capa gratuita, demostró ser una decisión acertada que garantiza la viabilidad económica del proyecto a largo plazo.

El proyecto se concibió y ejecutó bajo una metodología ágil, guiada estrictamente por historias de usuario, lo que facilitó un progreso iterativo e incremental. Si bien no se elaboró un diagrama de Gantt formal, este enfoque aseguró que cada funcionalidad desarrollada aportará un valor directo al usuario final, proporcionando un marco de trabajo flexible pero estructurado que dio un orden y una dirección clara a la implementación.

Aunque al momento de redactar este informe el sistema está completamente operativo, reconozco que el área de testing formal (pruebas unitarias y de integración) no fue un foco principal. Soy plenamente consciente de que, en un entorno de desarrollo profesional, esta es un área crucial para garantizar que el valor entregado sea robusto, confiable y mantenible a futuro.

En retrospectiva, considero que los objetivos planteados al inicio del proyecto fueron cumplidos satisfactoriamente. Se ha logrado una solución funcional y completa que no solo resuelve la problemática identificada de las largas filas en la cantina, sino que también se alinea con los principios de Responsabilidad Social Universitaria al mejorar la calidad de vida en el campus, promover la inclusión digital y generar un impacto medioambiental positivo. Este proyecto ha sido mucho más que un ejercicio técnico; me ha preparado para enfrentar proyectos tecnológicos reales con una visión más completa, estratégica e integral, entendiendo que la tecnología es una herramienta poderosa para generar valor y mejorar el entorno que nos rodea.

Próximos Pasos

Aunque el sistema actual es completamente funcional, el ciclo de vida de un producto de software nunca termina. Para asegurar su éxito y sostenibilidad a largo plazo, los siguientes pasos son fundamentales:

1. **Reforzar la Seguridad General:** Si bien se han implementado prácticas de seguridad estándar, como la gestión de secretos en GCP, un próximo paso crucial es realizar una auditoría de seguridad exhaustiva. Esto implicaría reforzar la seguridad de los endpoints de la API para prevenir vulnerabilidades comunes y asegurar la protección integral de los datos de los usuarios. Actualmente, **Firebase Authentication** se utiliza únicamente para vincular direcciones de correo electrónico con cuentas de usuario y así permitir guardar el historial de órdenes compradas. Como mejora futura, se propone que **el backend requiera y valide tokens de Firebase en los endpoints más sensibles** (por ejemplo, los relacionados con pagos, administración o modificaciones de datos críticos). Esto permitirá garantizar que solo usuarios autenticados puedan acceder a operaciones clave, elevando significativamente el nivel de seguridad general del sistema.

2. **Implementar Pruebas Automatizadas**: Para garantizar la robustez y facilitar el mantenimiento futuro, es prioritario desarrollar un conjunto completo de pruebas unitarias y de integración. Estas pruebas deben integrarse en el flujo de trabajo de CI/CD con herramientas como GitHub Actions, asegurando que cada nuevo cambio sea verificado automáticamente antes del despliegue y evitando regresiones en el código existente.

3. **Puesta en Producción y Validación Real**: El lanzamiento de la aplicación implica un proceso multifacético que va más allá del simple despliegue técnico. Este proceso incluye:

    - **Validación del Producto**: Permitir que los estudiantes y docentes utilicen la aplicación en su día a día. Esta etapa es crucial para obtener feedback directo y validar si la solución realmente cumple con sus expectativas y resuelve el problema de manera efectiva.

    - **Onboarding de la Cantina**: Realizar una capacitación con el personal administrativo de la cantina para asegurar que puedan utilizar la interfaz de administración de forma autónoma y eficiente (gestión de productos, visualización de órdenes, etc.). Este proceso de "onboarding" estaría sujeto a posibles modificaciones y mejoras, adaptando la herramienta a su flujo de trabajo real basándose en la retroalimentación de su experiencia de usuario (UX).

    - **Ciclo de Mejora Continua**: Utilizar los datos y comentarios recopilados durante la puesta en producción para iterar sobre el producto, corrigiendo errores, mejorando funcionalidades existentes y planificando futuras características.

4. **Optimización de Consumos e Imágenes en la Infraestructura:** Un siguiente paso clave será optimizar el uso de los recursos de infraestructura, especialmente los relacionados con la carga y entrega de imágenes en la aplicación web. Estas mejoras no solo disminuirán el consumo de ancho de banda y almacenamiento en

GCP, sino que también permitirán que Vercel aproveche su caché interna, evitando solicitudes repetidas de imágenes que no cambian frecuentemente, mejorando así la velocidad y eficiencia general del sistema.Para ello, se propone:

- **Optimización de imágenes**: Al subir nuevas imágenes desde el frontend convertirlas a formato WebP desde el backend antenas de subirlas al bucket, para reducir el peso de los archivos sin comprometer la calidad visual.

- **Mejorar la Configuración de Google Cloud Storage (GCS)**:

  - Establecer en el bucket de GCP encabezados de cacheo prolongado para imágenes inmutables

    ```
    Cache-Control: public, max-age=31536000, immutable
    ```

  - Implementar un sistema de versionado en los nombres de archivos para garantizar que los navegadores actualicen solo las imágenes modificadas.

  - Automatizar la eliminación de versiones antiguas al subir una nueva imagen, evitando almacenamiento innecesario y reduciendo costos.

5. **Implementar un Módulo de Reclamos y Reembolsos:** Desarrollar una nueva sección que permita a los usuarios (estudiantes y docentes, según lo solicitado en la encuesta) generar reclamos. Esto incluye la capacidad de realizar reclamos genéricos o vincular un reclamo a un order_id específico. El módulo podria contar con:

   - Una interfaz de administración para que el personal de la cantina pueda visualizar y gestionar todos los reclamos entrantes.

   - Funcionalidad de reembolsos: "Como administrador de la cantina quiero poder realizar reembolsos a través de la api de mercadopago, para poder reembolsar una compra por este medio si llegara a pasar algún inconveniente".

6. **Desarrollar un Módulo de Analíticas Avanzadas:** Crear un nuevo módulo de reportes: "Como administrador del sistema, quiero poder generar reportes de ventas y popularidad de productos, para entender mejor qué productos son los más vendidos y planificar futuras promociones o inventarios". Se evaluará la integración de herramientas de IA para potenciar estas analíticas.

7. **Integrar Funcionalidad de WhatsApp Business:** Incorporar la API de WhatsApp para mejorar la accesibilidad y permitir nuevos flujos de usuario:

   - Permitir a usuarios registrados iniciar sesión con su número de teléfono.

   - Habilitar compras para usuarios no registrados: El flujo permitiría confirmar el carrito, enviar el link de pago de Mercado Pago y recibir el resumen de la compra final a través de WhatsApp. El usuario podría retirar el pedido presentando el chat, sin necesidad de crear una cuenta.

8. **Implementar un Sistema de Puntos:** Diseñar e implementar un sistema de fidelización basado en puntos, cuya viabilidad y reglas de negocio deberán validarse con la administración de la cantina.

ANEXOS

1. **Entrevista con la Administración de la Cantina**
2. **Encuesta a usuarios de la cantina** - Para acceder deberá pedir acceso - También se puede ver el **excel con las respuestas** donde se pueden aplicar distintos filtros y ver los gráficos con los filtros aplicados
3. **Caso Gallina Blanca**
4. **Caso PIA**
5. **Google Datacenters**

BIBLIOGRAFÍA

1. Chat GPT por consultas y estructuración del texto
2. Google Cloud Run Pricing Savings Guide - Pump, https://www.pump.co/blog/google-cloud-run-pricing
3. Firestore pricing - Google Cloud, https://cloud.google.com/firestore/pricing
4. Understand Cloud Firestore billing | Firebase https://firebase.google.com/docs/firestore/pricing
5. GCP Storage Pricing - Cost Guide & Savings Strategies - Pump https://www.pump.co/blog/gcp-storage-pricing
6. Google Cloud Storage Pricing: Get the Best Bang for Your Buckets - NetApp BlueXP, https://bluexp.netapp.com/blog/gcp-cvo-blg-google-cloud-storage-pricing-get-the-best-bang-for-your-buckets
7. Cloud Run | Google Cloud https://cloud.google.com/run#pricing
8. Pricing | Cloud Run | Google Cloud https://cloud.google.com/run/pricing/
9. How much memory does a spring boot rest api usually consume? : r/SpringBoot - Reddit https://www.reddit.com/r/SpringBoot/comments/15gy5jy/how_much_memory_does_a_spring_boot_rest_api/
10. Memory management and garbage collection (GC) in ASP.NET Core - Learn Microsoft https://learn.microsoft.com/en-us/aspnet/core/performance/memory?view=aspnetcore-9.0
11. Pricing | Cloud Storage | Google Cloud https://cloud.google.com/storage/pricing
12. Free cloud features and trial offer | Google Cloud Free Program https://cloud.google.com/free/docs/free-cloud-features
13. Free Trial and Free Tier Services and Products - Google Cloud https://cloud.google.com/free
14. Google Cloud CDN Pricing & Savings Guide - Pump https://www.pump.co/blog/google-cloud-cdn-pricing
15. Fuentes sobre el papel: fundacioncanal.com - Mamá Coca - adoc Studio
16. Parrot Software. (n.d.). Pros y contras de las aplicaciones de delivery para restaurantes. Recuperado de https://parrotsoftware.com.mx/blog/pros-y-contras-de-las-aplicaciones-de-delivery-para-restaurantes
17. Ticksy. (n.d.). Ventajas y desventajas del servicio delivery. Recuperado de https://ticksy.app/blog/ventajas-y-desventajas-del-servicio-delivery/

18. ClickUp. (n.d.). Los 10 mejores programas de gestión alimentaria. Recuperado de https://clickup.com/es-ES/blog/436583/software-de-gestion-alimentaria

19. Vev. (n.d.). Los mejores software de entrega de comida a domicilio. Recuperado de https://vev.co/es/blog/los-mejores-software-de-entrega-de-comida-a-domicilio

20. Catalogo de Software. (n.d.). Software para restaurantes. Recuperado de https://www.catalogodesoftware.com/software/software-hospitales-pos-servicios-turismo-restaurantes/software-hoteles-bares-restaurantes-bogota-colombia/sistema-restaurantes-bogota-colombia-tns

21. Square. (n.d.). Software de Square KDS sin costo adicional. Recuperado de https://squareup.com/us/es/point-of-sale/restaurants

22. Aplyca. (n.d.). Next.js: El futuro web. Recuperado de https://www.aplyca.com/blog/nextjs-el-futuro-web-que-es-nextjs

23. 10code. (n.d.). ¿Qué Es Next.js y Cuál Es Su Propósito? Recuperado de https://10code.es/nextjs-que-es/

24. Solbyte. (n.d.). React JS: Ventajas e inconvenientes. Recuperado de https://www.solbyte.com/blog/react-js-ventajas-e-inconvenientes/

25. Serverspace. (n.d.). React.js: ventajas, desventajas y casos de uso. Recuperado de https://serverspace.io/es/about/blog/review-of-the-react-js-framework-advantages-disadvantages-and-use-cases/

26. The Codest. (n.d.). Pros y contras de Vue.js. Recuperado de https://thecodest.co/es/blog/pros-y-contras-de-vue/

27. Rootstack. (n.d.). VueJS: Ventajas y desventajas de este framework. Recuperado de https://rootstack.com/es/blog/vuejs-ventajas-desventajas

28. Discrat. (n.d.). Ventajas y Desventajas del responsive design. Recuperado de https://www.discrat.com.ar/ventajas-y-desventajas-del-diseno-adaptativo/

29. ESIC. (n.d.). Angular: Qué es, para qué sirve y ventajas. Recuperado de https://www.esic.edu/rethink/tecnologia/angular-que-es-para-que-sirve-y-ventajas-c

30. FastAPI. (n.d.). Características. Recuperado de https://fastapi.tiangolo.com/es/

31. IronPDF. (n.d.). FastAPI Python. Recuperado de https://ironpdf.com/es/python/blog/compare-to-other-components/fastapi-python/

32. Scribd. (n.d.). Django Rest Framework. Recuperado de https://es.scribd.com/document/538658165/Django-Rest-Framework

33. Django REST Framework. (n.d.). API Guide: Views. Recuperado de https://www.django-rest-framework.org/api-guide/views/

34. Apuntes.de. (n.d.). Desarrollo de una API REST con Flask en Python. Recuperado de https://apuntes.de/python/desarrollo-de-una-api-rest-con-flask-en-python-creacion-de-una-interfaz-de-programacion-de-aplicaciones-restful/

35. Certidevs. (n.d.). Tutorial Flask API REST GET. Recuperado de https://certidevs.com/tutorial-flask-api-rest-get

36. Hostinger. (n.d.). ¿Qué es Node.js? Recuperado de https://www.hostinger.com/es/tutoriales/que-es-node-js

37. Startechup. (n.d.). Node.js vs Express.js: Características y Ventajas. Recuperado de https://www.startechup.com/es/blog/tospanish/

38. Geekflare. (n.d.). AWS vs. Azure vs. Google Cloud. Recuperado de https://geekflare.com/es/aws-vs-azure-vs-google-cloud/

39. Docker. (n.d.). Docker Hub. Recuperado de https://www.docker.com/products/docker-hub/

40. Palo Alto Networks. (n.d.). ¿Qué es el ciclo de CI/CD? Recuperado de https://www.paloaltonetworks.es/cyberpedia/what-is-the-ci-cd-pipeline-and-ci-cd-security

41. GitHub. (n.d.). What is CI/CD? Recuperado de https://github.com/resources/articles/devops/ci-cd

42. GitHub. (n.d.). Getting started with GitHub Actions. Recuperado de https://docs.github.com/articles/getting-started-with-github-actions