SISTEMA DE GESTIÓN DE PEDIDOS

Software de administración empresarial

Videla Guliotti, Tobías Uriel

Universidad Nacional de Villa Mercedes
Programador Universitario de Sistemas
Práctica Profesional Supervisada

1 de noviembre de 2025

Índice General

ÍNDICE GENERAL

Introducción

El presente documento tiene como objetivo documentar el desarrollo de un sistema de información destinado a la administración de pedidos, clientes, proveedores y productos. Se busca analizar las alternativas disponibles para abordar el problema, fundamentar las decisiones tomadas y proponer una solución de base que pueda adaptarse a distintas organizaciones con necesidades similares.

El trabajo se origina a partir de un proyecto real orientado a digitalizar y optimizar procesos operativos de un emprendimiento local dedicado a la comercialización de productos agroecológicos y orgánicos. Si bien el problema concreto surge de una situación particular, el diseño del sistema pretende ser lo suficientemente general como para resultar aplicable a otros modelos de negocio similares.

A lo largo de este documento se realizará el planteamiento del problema, el análisis de metodologías de desarrollo de software, la utilización de herramientas de modelado como UML, la evaluación de tecnologías de implementación y, finalmente, la exposición de una propuesta de solución.

Capítulo 1

Planteo del Problema

El emprendimiento local, en el que se basa el planteo del problema a resolver, se dedica a la distribución de productos agroecológicos y orgánicos, incluyendo frutas, verduras y artículos de almacén. Este emprendimiento recibe semanalmente pedidos de múltiples clientes minoristas, así como de un cliente mayorista.

La operatoria del negocio presenta una serie de características particulares. Por un lado, los proveedores del emprendimiento modifican sus listas de precios y productos cada semana, lo que obliga a actualizar constantemente la información comercial. Por otro lado, el cálculo de los precios finales requiere la aplicación de distintos porcentajes de ganancia por producto según el tipo de cliente (mayorista o minorista), lo que representa una tarea repetitiva y propensa a errores si se realiza de forma manual.

Hasta el momento, la gestión se lleva a cabo mediante planillas de cálculo (Excel) que el cliente adapta semanalmente para calcular precios, organizar pedidos y realizar compras. Uno de los proveedores le envía una hoja de Excel para completar con los productos solicitados, mientras que otro proveedor ha comenzado recientemente a utilizar una plataforma web con carrito de compras.

Este modelo de trabajo manual y disperso presenta diversas limitaciones: requiere una importante dedicación de tiempo, implica una alta carga mental, y dificulta la trazabilidad y sistematización de los pedidos. Además, la constante necesidad de adaptación frente a cambios semanales genera un desgaste que podría mitigarse mediante herramientas digitales específicas.

En este contexto, surge la necesidad de desarrollar un sistema que automatice la mayor cantidad de tareas posibles, simplifique aquellas que deban seguir realizándose de forma manual y proporcione una interfaz intuitiva y eficiente para el usuario. El presente documento se orienta a analizar cuál es la mejor metodología de desarrollo para abordar el diseño y la construcción de dicha solución.

1.1. Requerimientos del Sistema

A partir del análisis del problema expuesto en la sección anterior, y con el objetivo de desarrollar un sistema de información integral y eficiente, adaptable a empresas con modelos de negocio similares, se definen a continuación los requerimientos funcionales y no funcionales que orientarán el diseño y la implementación de la solución propuesta.

1.1.1. Requerimientos Funcionales

- **Gestión de Productos:**
  - Alta, baja, modificación y consulta de productos.
  - Atributos de productos: nombre, origen, productor, peso (kg), descripción, precio de compra, precio de venta minorista, precio de venta mayorista, precio por kilo, cantidad disponible (stock) y categoría.
- **Gestión de Listas de Productos:**
  - Creación de listas de productos ofrecidos, agrupadas por categorías.
  - Creación de sublistas o agrupaciones (mixes) dentro de cada lista.
  - Generación de una lista general que consolide todos los productos disponibles.
  - Posibilidad de exportar listas (individuales o generales) a formato PDF, permitiendo seleccionar qué detalles incluir (precios, descripciones, cantidades, etc.).
- **Gestión de Clientes:**
  - Alta, baja, modificación y consulta de clientes minoristas y mayoristas.
  - Registro de datos: nombre, dirección, teléfono, correo electrónico, tipo de cliente (minorista/mayorista) y estado de actividad.
- **Gestión de Proveedores:**
  - Alta, baja, modificación y consulta de proveedores.
  - Registro de datos: nombre, teléfono, correo electrónico, y lista de productos suministrados.
- **Gestión de Stock y Pedidos a Proveedores:**
  - Carga de productos al stock mediante archivos (Excel, factura, etc.).
  - Asociación de productos ingresados a un proveedor específico.
  - Confirmación de compras o generación automática de pedidos a proveedores (vía correo electrónico o WhatsApp).

  - Actualización automática del stock tras confirmar pedidos.

- **Plataforma Web para Clientes:**
  - Acceso mediante nombre de usuario y contraseña (previo registro en base de datos).
  - Visualización de productos con precios diferenciados según el tipo de cliente.
  - Funcionalidad de carrito de compras para realizar pedidos.
- **Gestión de Pedidos de Clientes:**
  - Visualización de pedidos recibidos, incluyendo datos del cliente y detalles del pedido.
  - Actualización y gestión del estado de cada pedido (preparación, envío, completado).
- **Estadísticas y Reportes:** Visualización de métricas relevantes para la toma de decisiones empresariales. La presentación de estas métricas deberá realizarse mediante gráficos, tablas o reportes exportables, brindando una visión clara y accesible de la situación operativa y comercial de la empresa.
  - **Resumen de ventas:** cantidad de pedidos concretados por mes, monto total de ventas mensuales, y ventas discriminadas por tipo de cliente (minorista o mayorista).
  - **Productos más vendidos:** identificación de los productos más solicitados en un período determinado, tanto en unidades como en ingresos generados.
  - **Clientes más activos:** listado de los clientes que han generado mayor volumen de compras en el mes o en un período seleccionado.
  - **Stock crítico:** productos cuyo stock disponible se encuentra por debajo de un umbral mínimo predefinido, permitiendo alertar sobre la necesidad de reposición.
  - **Análisis de proveedores:** evaluación de la frecuencia de compras y volumen de productos adquiridos a cada proveedor, facilitando la gestión de relaciones comerciales.
  - **Comparativas temporales:** análisis de la evolución de ventas, stock y pedidos a lo largo de distintos meses o años.

1.1.2. Requerimientos No Funcionales

- **Usabilidad:** El sistema deberá contar con una interfaz intuitiva y amigable tanto para usuarios administrativos como para clientes.

- **Seguridad:** Implementación de mecanismos de autenticación y control de acceso para clientes y administradores.
- **Portabilidad:** La plataforma web deberá ser accesible desde dispositivos móviles y computadoras de escritorio.
- **Rendimiento:** El sistema deberá ser capaz de gestionar múltiples pedidos y operaciones de manera eficiente, incluso ante incrementos en el volumen de datos.
- **Mantenibilidad:** El sistema deberá estar diseñado siguiendo buenas prácticas de programación, permitiendo su actualización y extensión futura.
- **Escalabilidad:** Deberá poder adaptarse al crecimiento del número de productos, clientes y proveedores sin afectar el rendimiento general.
- **Confiabilidad:** El sistema debe garantizar la correcta persistencia y recuperación de datos en todo momento.

Capítulo 2

Marco Teórico y Estado del Arte

2.1. Estado del Arte

Actualmente, los pequeños emprendimientos agroecológicos y de venta directa de productos orgánicos recurren a diversas herramientas tecnológicas para gestionar sus operaciones. En la mayoría de los casos, la administración de pedidos, cálculo de precios, control de stock y contacto con proveedores se realiza mediante planillas de cálculo, principalmente utilizando Microsoft Excel o Google Sheets. Estas soluciones ofrecen flexibilidad y bajo costo, pero también presentan limitaciones en cuanto a escalabilidad, automatización y facilidad de uso.

Por otra parte, existen sistemas más robustos como los software de gestión empresarial (ERP), que integran módulos para ventas, compras, inventario, contabilidad, y más. Sin embargo, estas herramientas suelen estar diseñadas para empresas medianas o grandes, y resultan complejas, costosas o sobredimensionadas para emprendimientos de menor escala. Además, requieren una curva de aprendizaje considerable y, en muchos casos, conocimientos técnicos para su implementación y mantenimiento.

En los últimos años, han surgido soluciones digitales específicas para el rubro alimenticio, como aplicaciones móviles y plataformas web para la gestión de pedidos, pero estas herramientas suelen estar orientadas a restaurantes, mercados a gran escala, o franquicias. En general, no contemplan particularidades del comercio agroecológico ni permiten una adaptación flexible a cambios semanales en precios o productos disponibles.

A nivel metodológico, el desarrollo de sistemas a medida para este tipo de emprendimientos no suele documentarse ampliamente, y se basa en enfoques prácticos adaptados a cada caso. La falta de documentación formal sobre proyectos similares plantea un desafío adicional, pero a la vez brinda la oportunidad de generar una solución novedosa, adecuada al contexto y con posibilidad de replicabilidad en otros emprendimientos con características similares.

En este contexto, resulta fundamental analizar las metodologías de desarrollo de soft-

ware disponibles, con el fin de elegir aquella que mejor se adapte a los objetivos, el tiempo disponible, los recursos humanos involucrados y la evolución progresiva del sistema.

2.2. Metodologías de Desarrollo de Software

En el ámbito del desarrollo de software, la elección de una metodología adecuada es un factor clave para garantizar la calidad del producto final, la eficiencia del proceso y la satisfacción de los usuarios. Las metodologías de desarrollo definen el conjunto de prácticas, principios y pasos a seguir a lo largo del ciclo de vida del software. A lo largo de la historia de la ingeniería de software han surgido diversas metodologías, que se pueden agrupar principalmente en dos enfoques: tradicionales y ágiles. En esta sección se describen ambas categorías, sus características, ventajas y desventajas, a fin de proporcionar el marco conceptual necesario para justificar la metodología seleccionada para el proyecto en curso.

2.2.1. Metodologías Tradicionales

Las metodologías tradicionales, también conocidas como metodologías planificadas o pesadas, están basadas en un enfoque secuencial y riguroso. Suelen dividir el proceso de desarrollo en etapas bien definidas y estructuradas que deben completarse en un orden específico. Estas metodologías fueron ampliamente utilizadas durante décadas, especialmente en proyectos de gran envergadura o con requerimientos estables desde el inicio.

Modelo en Cascada

El modelo en cascada es uno de los enfoques más conocidos dentro de las metodologías tradicionales. Este modelo propone un flujo lineal de actividades: análisis de requerimientos, diseño, implementación, pruebas, despliegue y mantenimiento. Cada etapa depende de la finalización de la anterior, lo que otorga un alto grado de control y documentación. Sin embargo, su principal desventaja es la rigidez ante cambios en los requerimientos, lo que lo vuelve poco adecuado para entornos dinámicos o inciertos.

Modelo Incremental

El modelo incremental plantea un enfoque en el que el sistema se desarrolla y entrega en partes o incrementos funcionales. Cada incremento incluye funcionalidades completas que se integran gradualmente hasta formar el sistema final. Este modelo permite entregar versiones utilizables del producto en etapas tempranas, lo que facilita la retroalimentación temprana por parte del usuario.

Entre sus ventajas se destacan la flexibilidad para incorporar cambios entre incrementos, la detección temprana de errores y la entrega continua de valor. No obstante, requiere

una planificación rigurosa para asegurar que los incrementos se integren armónicamente, y puede no ser adecuado si el sistema completo debe ser entregado en una única fase.

Modelo Espiral

El modelo espiral combina elementos del modelo en cascada con técnicas de análisis de riesgos y prototipado. Cada iteración del espiral representa un ciclo completo del desarrollo, donde se evalúan riesgos, se construyen prototipos y se obtiene retroalimentación. Si bien es más flexible que los modelos anteriores, su complejidad lo hace menos accesible para proyectos pequeños o equipos con poca experiencia.

2.2.2. Metodologías Ágiles

Frente a la rigidez de los enfoques tradicionales, surgieron las metodologías ágiles como una alternativa más flexible y centrada en las personas. Estas metodologías promueven la entrega continua de valor, la adaptación al cambio y la colaboración entre los distintos actores del proyecto. El desarrollo ágil se basa en el *Manifiesto Ágil* (2001)[1], que resalta la importancia de individuos e interacciones, software funcionando, colaboración con el cliente y respuesta ante el cambio.

Scrum

Scrum es una de las metodologías ágiles más populares. Organiza el trabajo en iteraciones llamadas *sprints*, que suelen durar entre una y cuatro semanas. Durante cada sprint, el equipo trabaja en un conjunto de funcionalidades priorizadas desde un *Product Backlog*. Scrum define roles específicos (Product Owner, Scrum Master, Development Team) y eventos regulares como reuniones diarias (*Daily Scrum*) y revisiones al final de cada sprint. Su enfoque iterativo e incremental permite una entrega continua de valor y una rápida adaptación a los cambios.

Kanban

Kanban es una metodología visual que busca mejorar el flujo de trabajo mediante un sistema de tarjetas que representan las tareas y un tablero dividido en columnas que reflejan los estados del proceso (por ejemplo: Por hacer, En progreso, Terminado). Su simplicidad y flexibilidad la hacen ideal para equipos que buscan optimizar su productividad sin la estructura formal de otras metodologías ágiles.

Extreme Programming (XP)

Extreme Programming se centra en mejorar la calidad del software y la capacidad de respuesta ante cambios a través de prácticas como la programación en parejas (*pair*

programming), el desarrollo guiado por pruebas (*Test Driven Development*), la integración continua y la retroalimentación frecuente del cliente. XP es especialmente útil en proyectos donde los requerimientos cambian constantemente y se requiere una alta calidad técnica.

2.2.3. Comparación General

En general, las metodologías ágiles se caracterizan por su adaptabilidad, entregas frecuentes y un enfoque constante en el usuario final. Son especialmente eficaces en proyectos con requerimientos cambiantes o cuando se busca involucrar activamente al cliente en el proceso de desarrollo. En contrapartida, pueden presentar dificultades en entornos altamente regulados, donde se requiere una planificación detallada desde el inicio, o en proyectos donde la disponibilidad del cliente es limitada.

En la actualidad, las metodologías ágiles dominan la escena del desarrollo de software, especialmente en startups, empresas tecnológicas y proyectos orientados al cliente. Su capacidad de adaptación y entrega continua de valor responde mejor a la dinámica del mercado actual, caracterizado por la incertidumbre y la necesidad de innovación constante. Aun así, las metodologías tradicionales continúan siendo utilizadas en sectores donde la planificación rigurosa es fundamental, como la industria aeroespacial o automotriz.

Finalmente, puede decirse que la principal fortaleza de las metodologías tradicionales reside en su capacidad de control, documentación y previsibilidad, lo cual las hace apropiadas para proyectos con requerimientos estables y bien definidos desde el inicio. En cambio, la mayor virtud de las metodologías ágiles es su flexibilidad y enfoque iterativo, lo que permite incorporar mejoras de forma continua y responder eficazmente al cambio.

Capítulo 3

UML - Lenguaje Unificado de Modelado

3.1. Introducción a UML

El Lenguaje Unificado de Modelado[2] (UML, por sus siglas en inglés: *Unified Modeling Language*) es un lenguaje gráfico utilizado para visualizar, especificar, construir y documentar los artefactos de un sistema software. UML proporciona una notación estandarizada y ampliamente aceptada para representar distintos aspectos del diseño y comportamiento de un sistema, lo cual facilita la comunicación entre los distintos actores involucrados en su desarrollo: analistas, diseñadores, desarrolladores y clientes.

El uso de UML permite abstraer y entender la estructura del sistema de manera clara y precisa, sin necesidad de escribir código desde el inicio. Mediante diagramas, se pueden modelar clases, objetos, procesos, flujos de datos, casos de uso, interacciones entre componentes y otros elementos fundamentales del sistema. Esto contribuye a detectar errores conceptuales antes de la implementación, favorece la modularidad del desarrollo y mejora la mantenibilidad del software.

Dado que el proyecto propuesto busca automatizar procesos de gestión para un emprendimiento que actualmente depende de hojas de cálculo y procedimientos manuales, el uso de UML resulta especialmente útil para representar de forma visual los distintos módulos del sistema, sus interacciones y los flujos de información. Además, permitirá establecer una base sólida para el desarrollo iterativo y la posterior documentación del producto final.

3.2. Diagramas de UML

UML organiza sus diagramas en tres grandes categorías según su propósito:

- **Diagramas Estructurales:** Modelan los elementos estáticos del sistema, como

clases, objetos y componentes, mostrando la organización y relaciones internas.

- **Diagramas de Comportamiento:** Describen el comportamiento dinámico del sistema, incluyendo flujos de actividades, cambios de estado y funcionalidades observables.
- **Diagramas de Interacción:** Son un subconjunto de los diagramas de comportamiento, centrados específicamente en el flujo de mensajes entre objetos y componentes durante las interacciones.

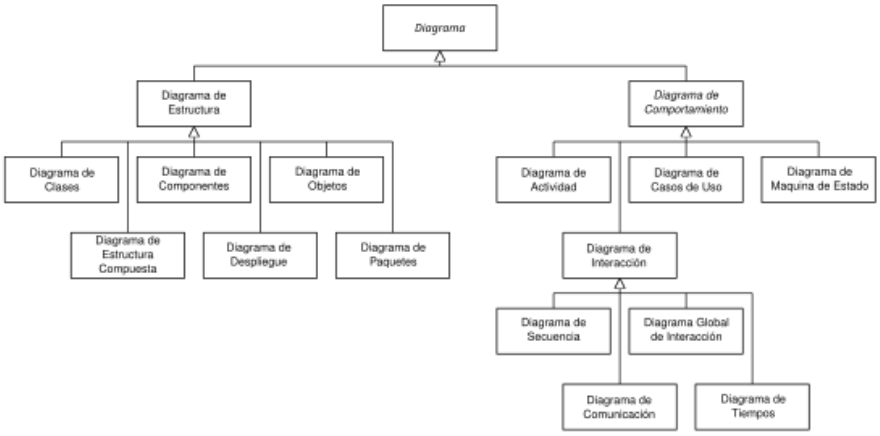


Figura 3.1: Clasificación de diagramas UML.

A continuación, se presentan brevemente los tipos de diagramas definidos en UML relevantes para este trabajo.

3.2.1. Diagrama de Clases

Representa las clases del sistema, sus atributos, operaciones y las relaciones entre ellas. Es fundamental para modelar la estructura lógica del sistema.

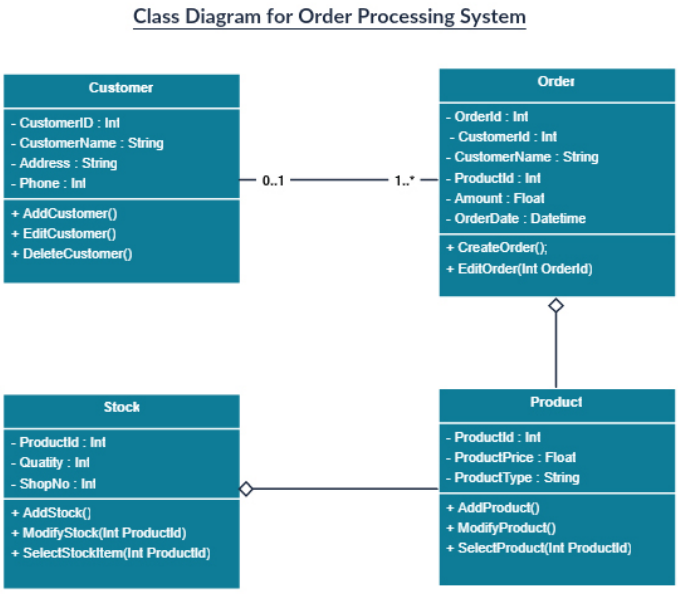


Figura 3.2: Ejemplo de Diagrama de Clases.[3]

3.2.2. Diagrama de Casos de Uso

Describe las funcionalidades que el sistema ofrece a los usuarios (actores) y cómo interactúan con él.

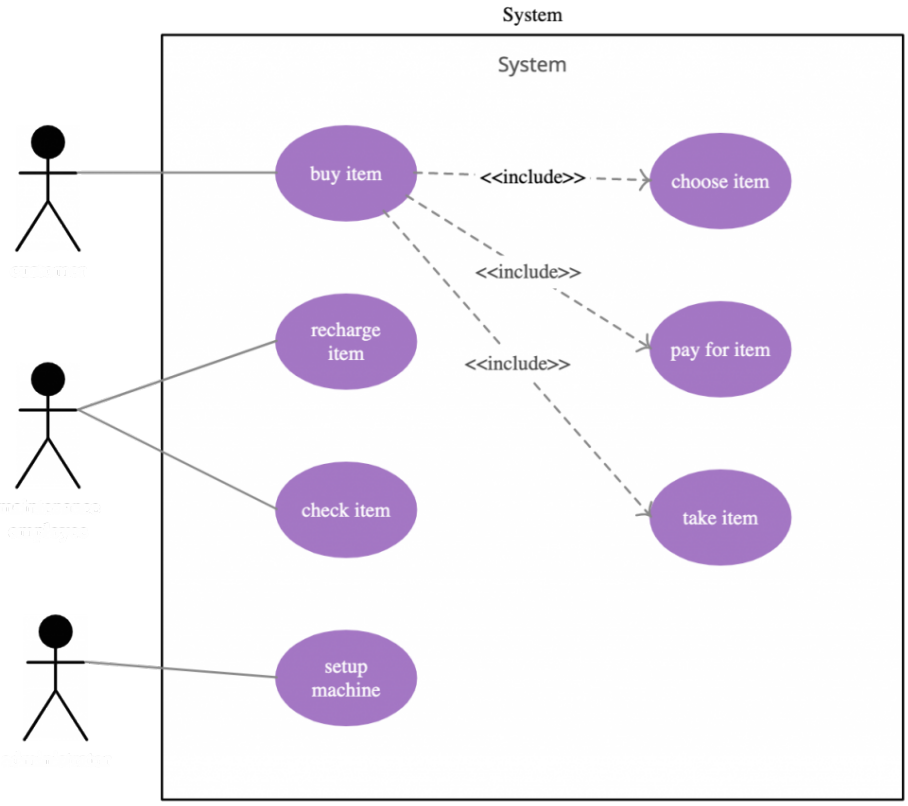


Figura 3.3: Ejemplo de Diagrama de Casos de uso.[3]

3.2.3. Diagrama de Secuencia

Muestra cómo los objetos interactúan en una secuencia temporal, destacando el orden de los mensajes.

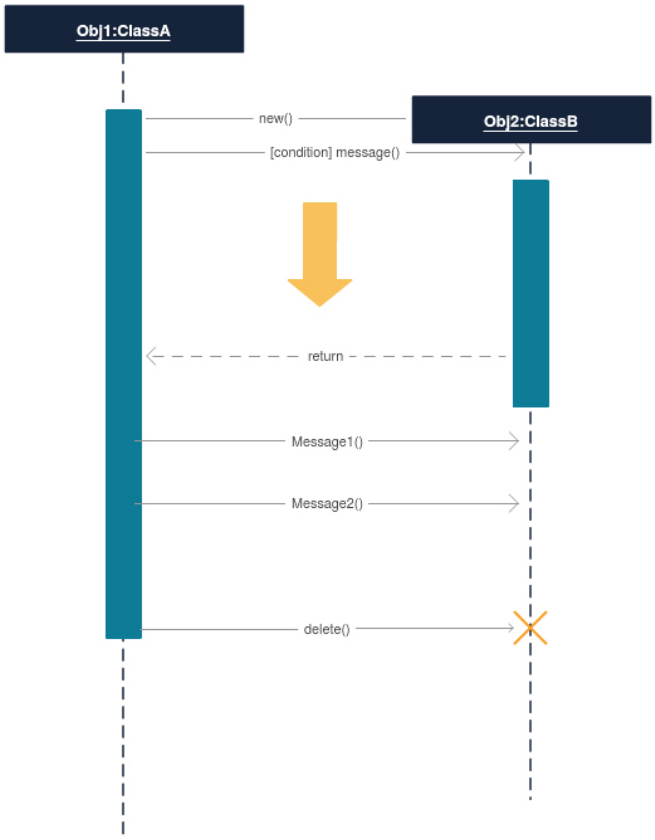


Figura 3.4: Ejemplo de Diagrama de Secuencia.[3]

3.2.4. Diagrama de Colaboración (o de Comunicación)

Similar al diagrama de secuencia, pero enfocado en la estructura de la comunicación más que en el tiempo.

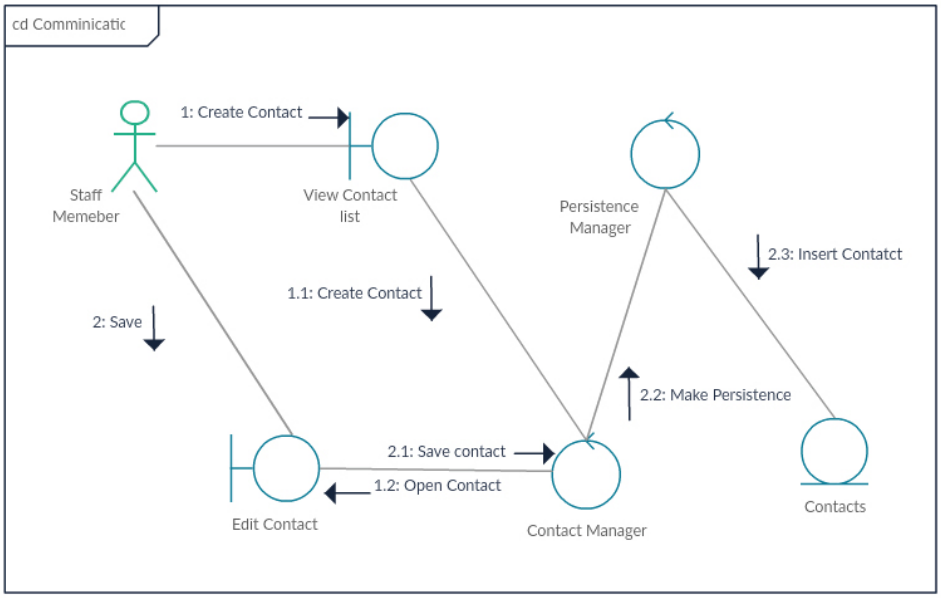


Figura 3.5: Ejemplo de Diagrama de Colaboración o Comunicación.[3]

3.2.5. Diagrama de Actividades

Modela flujos de trabajo o procesos, mostrando las actividades y las decisiones dentro del sistema.

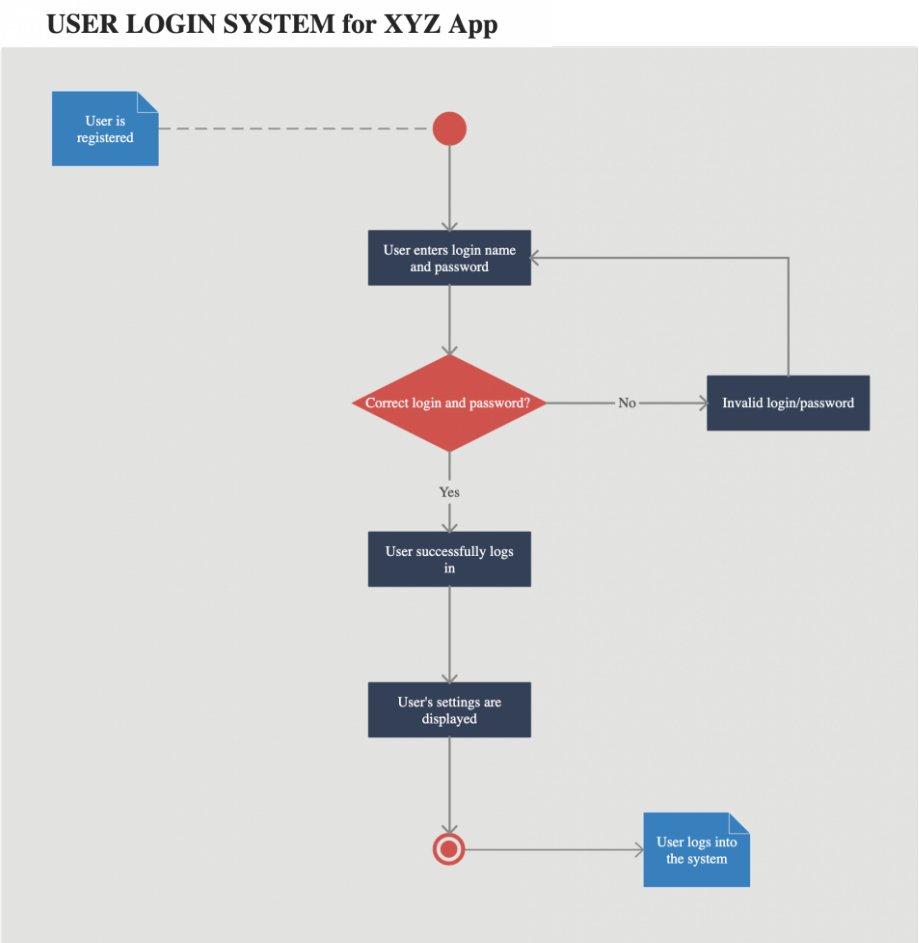


Figura 3.6: Ejemplo de Diagrama de Actividad[3]

Capítulo 4

Lenguajes y Tecnologías para la Implementación

4.1. Introducción

En esta sección se analizan diversos lenguajes de programación y tecnologías disponibles que podrían ser utilizados para la implementación del sistema. La elección adecuada de las herramientas tecnológicas resulta fundamental para garantizar la eficiencia, escalabilidad y facilidad de mantenimiento del proyecto.

4.2. Lenguajes de Programación

Entre los lenguajes de programación considerados para el desarrollo del sistema, se destacan los siguientes:

4.2.1. Python

Python es un lenguaje de alto nivel, multiparadigma y con una sintaxis clara y concisa. Presenta una amplia comunidad de soporte y un vasto ecosistema de librerías, lo cual facilita el desarrollo ágil de aplicaciones. Es especialmente apto para proyectos que requieren automatización, procesamiento de datos o desarrollo web rápido mediante frameworks como Django o Flask.

4.2.2. JavaScript

JavaScript es el lenguaje por excelencia para el desarrollo del lado del cliente en aplicaciones web. Además, mediante entornos como Node.js, también puede utilizarse en el backend, permitiendo una arquitectura basada en un único lenguaje en todo el proyecto.

4.2.3. Java

Java es un lenguaje orientado a objetos, maduro y ampliamente utilizado en el desarrollo de aplicaciones empresariales. Su portabilidad, gracias a la máquina virtual Java (JVM), y su robustez, lo convierten en una opción sólida para sistemas que requieran alta fiabilidad.

4.3. Bases de Datos

Para la persistencia de datos, se contemplan distintas opciones de sistemas de gestión de bases de datos (SGBD):

4.3.1. MySQL

MySQL es uno de los SGBD relacionales más populares, ideal para aplicaciones donde se requiere integridad y estructuración de datos. Es ampliamente soportado por numerosos lenguajes y plataformas.

4.3.2. PostgreSQL

PostgreSQL es un SGBD relacional avanzado que soporta operaciones complejas y es altamente extensible. Es especialmente recomendable para sistemas que requieran alta consistencia, operaciones complejas o bases de datos de gran tamaño.

4.3.3. SQLite

SQLite es una opción ligera y embebida, ideal para aplicaciones que no requieran un servidor de base de datos separado, como aplicaciones de escritorio o móviles.

4.4. Frameworks y Herramientas de Desarrollo

En el proceso de desarrollo del sistema propuesto, resulta fundamental apoyarse en un conjunto de herramientas que no solo agilicen la programación, sino que también garanticen un trabajo colaborativo eficiente, control de versiones, despliegue del sistema y mantenimiento a largo plazo.

4.4.1. Django

Framework de alto nivel para Python que permite un desarrollo rápido y limpio de aplicaciones web. Incluye funcionalidades integradas para la gestión de usuarios, bases de datos y administración de contenidos.

4.4.2. Flask

Framework minimalista para Python que proporciona flexibilidad para construir aplicaciones web ligeras y modulares.

4.4.3. React

Librería de JavaScript desarrollada por Facebook, ideal para construir interfaces de usuario dinámicas y reactivas.

4.4.4. Bootstrap

Framework de CSS que facilita la creación de interfaces web adaptables y profesionales de manera rápida.

4.4.5. Git y GitHub

Una de las herramientas más utilizadas a nivel mundial para el control de versiones es **Git**, un sistema distribuido que permite llevar un registro detallado de los cambios realizados en el código fuente. Su uso facilita el trabajo colaborativo entre desarrolladores, permitiendo ramificar el desarrollo, fusionar funcionalidades, y deshacer cambios en caso de errores.

GitHub es una plataforma para alojar repositorios remotos y coordinar el trabajo en equipo. Ofrece funcionalidades como pull requests, revisión de código, seguimiento de issues y automatización de tareas mediante GitHub Actions. Estas características la convierten en una herramienta clave para la organización y profesionalización del ciclo de desarrollo.

4.4.6. Servidores Web

Una vez desarrollada la solución, será necesario desplegar el sistema en un entorno accesible para los usuarios finales. Para este propósito, se puede optar por levantar un servidor web utilizando herramientas como:

- **Apache HTTP Server**: uno de los servidores web más utilizados, compatible con una gran variedad de tecnologías.
- **Nginx**: alternativa moderna y liviana, muy eficiente para manejar gran cantidad de conexiones simultáneas y balanceo de carga.
- **Gunicorn** (en caso de aplicaciones Python/Flask/Django): servidor WSGI que puede combinarse con Nginx para mayor eficiencia.

4.4.7. Plataforma de Hosting

En cuanto al alojamiento, existen múltiples alternativas según las necesidades y presupuesto del proyecto. Algunas opciones son:

- **Heroku**: plataforma como servicio (PaaS) que permite desplegar aplicaciones web fácilmente sin necesidad de gestionar infraestructura.
- **Render**: alternativa moderna a Heroku con soporte para backend, frontend, y bases de datos.
- **Vercel** o **Netlify**: ideales para alojar frontends modernos (React, Vue, etc.), integradas con GitHub para despliegues automáticos.

La elección de la plataforma dependerá de factores como el lenguaje de programación utilizado, la complejidad del sistema, el presupuesto disponible y el grado de control deseado sobre la infraestructura.

Capítulo 5

Conclusiones y Planteo de la Solución

A lo largo del presente trabajo se ha desarrollado un análisis integral del problema a resolver, se exploraron metodologías de desarrollo tradicionales, se evaluaron lenguajes y tecnologías disponibles, y se propuso un modelo estructurado para abordar el diseño de un sistema de información destinado a la gestión de productos, clientes, proveedores y pedidos.

En función del contexto planteado —un equipo de trabajo pequeño, con tiempos acotados y la necesidad de establecer una base sólida y mantenible— se opta por adoptar una metodología tradicional de desarrollo, similar al modelo en cascada. Esta elección responde a su simplicidad, claridad en las etapas y facilidad de organización del trabajo, además de su escalabilidad para futuros avances o incorporaciones de equipo. La metodología elegida permite enfocarse primero en el diseño detallado del sistema antes de pasar a su implementación, lo que favorece la comprensión profunda del problema y la alineación con los requerimientos funcionales y no funcionales previamente definidos.

Para la implementación futura del sistema se seleccionaron tecnologías ampliamente utilizadas y con gran soporte en la comunidad, tales como:

- **Python**, como lenguaje de programación principal para el backend del sistema.
- **PostgreSQL**, como sistema gestor de bases de datos relacional.
- **JavaScript**, **HTML** y **CSS**, para la construcción de la interfaz web del lado del cliente.
- **Bootstrap**, como framework para diseño responsivo y ágil de la interfaz de usuario.
- **Docker**, como herramienta para la containerización de aplicaciones, facilitando la creación de entornos aislados, portables y reproducibles tanto en desarrollo como en producción.
- **Git** y **GitHub**, como herramientas de control de versiones y trabajo colaborativo.

En cuanto al diseño del sistema, se empleará el lenguaje de modelado UML para representar las distintas perspectivas del mismo. Se prevé la elaboración de los siguientes diagramas: de **Casos de Uso**, **Clases**, **Actividad**, **Secuencia**, y **Colaboración o Comunicación**. Además, se utilizarán **Tarjetas CRC (Clase-Responsabilidad-Colaborador)** como herramienta complementaria para refinar el diseño de clases.

Dado el tiempo limitado disponible para esta etapa del proyecto, se abordará parcialmente la solución, concentrándose en el desarrollo de los diagramas correspondientes a la operación de **gestión de pedidos** desde la plataforma web para clientes, así como su recepción y procesamiento en el sistema local. Esta funcionalidad representa uno de los ejes centrales del sistema y permitirá sentar las bases para futuras extensiones.

Finalmente, se diseñará una **base de datos inicial y simplificada**, que servirá como punto de partida para el desarrollo de un prototipo funcional en etapas posteriores. Esta aproximación permite validar tempranamente el diseño propuesto, facilitando su evaluación, mejora y evolución futura.

5.1. Casos de Uso

En esta sección se presentarán los casos de uso. Primeramente el caso de uso general, seguido de los casos de uso específicos que se tratarán en la presente obra.

5.1.1. General

El sistema propuesto se estructura en torno a la gestión integral de pedidos, permitiendo a los clientes realizar sus pedidos desde una plataforma web y al administrador gestionar dichos pedidos desde un sistema local. Los casos de uso están organizados en paquetes que representan los distintos módulos funcionales: la **Plataforma Web** y el **Sistema de Información Local**.

Además, se contemplan acciones automatizadas que el sistema ejecuta como parte del flujo normal de operación.

Los actores principales identificados son:

- **Cliente** (dividido internamente en minorista y mayorista según el tipo de precio que se le muestra).
- **Administrador**, responsable de validar y gestionar pedidos.
- **Sistema de Información**, encargado de automatizar procesos clave (como actualización de stock o envío de pedidos a proveedores).
- **Proveedor**, actor externo que provee listas de precios y recibe los pedidos correspondientes.

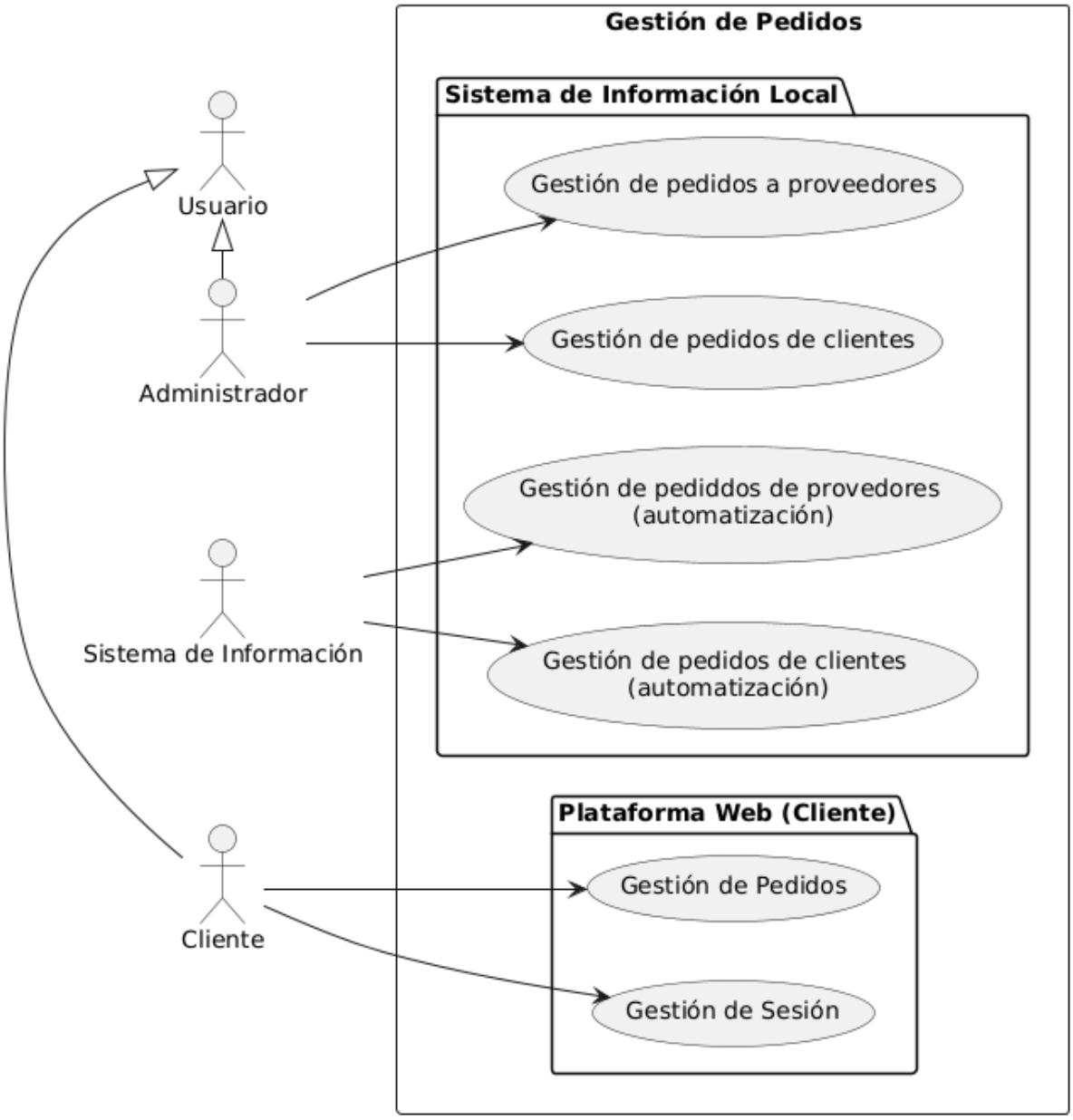


Figura 5.1: Diagrama de Casos de Uso - Gestión de Pedidos

5.1.2. Plataforma Web

La **Plataforma Web** está diseñada para que los clientes puedan interactuar de manera sencilla con el catálogo de productos y realizar sus pedidos.

Los casos de uso contemplados para este módulo incluyen:

- **Iniciar sesión / Cerrar sesión**: Permite al cliente autenticarse en la plataforma y finalizar su sesión de forma segura.
- **Buscar productos**: El cliente puede navegar y filtrar el catálogo de productos disponibles.

- **Agregar productos al carrito**: Permite seleccionar productos deseados y guardarlos temporalmente antes de confirmar la compra.
- **Realizar pedido**: Al completar la selección de productos, el cliente puede enviar su pedido, que será registrado en el sistema.

Estos casos de uso están contenidos dentro del paquete **"Plataforma Web (Cliente)"**, y son ejecutados exclusivamente por el actor **Cliente**.

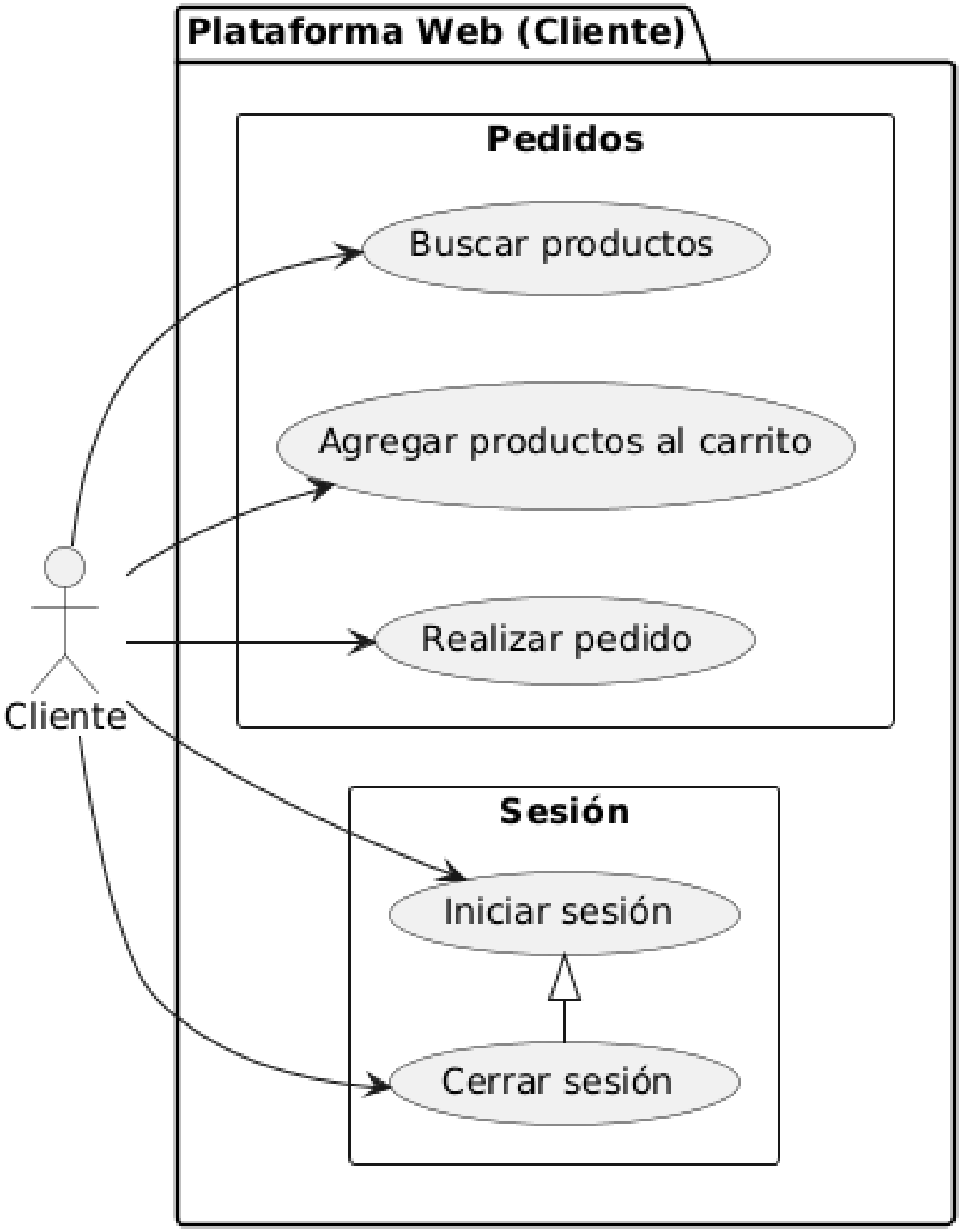


Figura 5.2: Diagrama de Casos de Uso - Plataforma Web (Cliente)

5.1.3. Sistema Local

El **Sistema de Información Local** está diseñado para ser operado por el administrador del sistema y automatizar ciertas tareas de gestión.

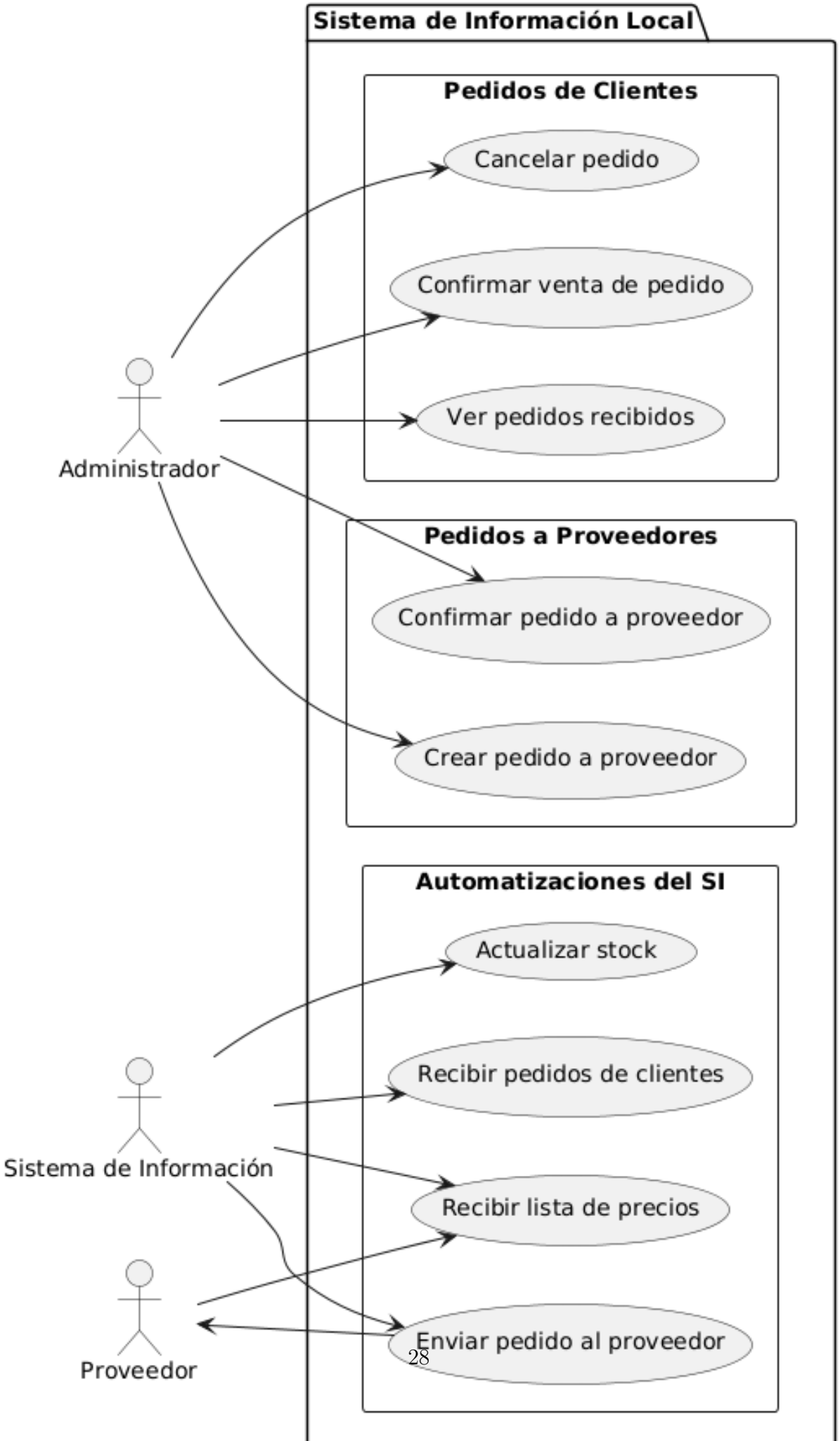
Los casos de uso para el **Administrador** incluyen:

- **Gestionar pedidos de clientes**: Permite visualizar, confirmar, cancelar o actualizar el estado de los pedidos realizados por los clientes.
- **Confirmar venta**: Acciones administrativas sobre el cierre de la venta.
- **Gestionar pedidos a proveedores**: Creación y confirmación de pedidos que se derivan al proveedor.

Los **casos de uso automatizados por el sistema** incluyen:

- **Recibir pedidos de clientes**: El sistema interpreta y almacena automáticamente los pedidos que llegan desde la plataforma web.
- **Actualizar stock**: Cada vez que se confirma una venta o se recibe un pedido del proveedor, el sistema actualiza el inventario de forma automática.
- **Enviar pedido al proveedor**: El sistema puede gestionar el envío de pedidos a través de canales como email o WhatsApp.
- **Recibir lista de precios del proveedor**: El sistema procesa listas de precios recibidas para actualizar la base de datos interna de productos.

Este conjunto de funcionalidades se encapsula en el paquete **"Sistema de Información Local"**, e involucra tanto acciones manuales (por el administrador) como automáticas (por el sistema).


Sistema de Información Local
Pedidos de Clientes
Cancelar pedido
Confirmar venta de pedido
Ver pedidos recibidos
Administrador
Pedidos a Proveedores
Confirmar pedido a proveedor
Crear pedido a proveedor
Automatizaciones del SI
Actualizar stock
Recibir pedidos de clientes
Sistema de Información
Recibir lista de precios
Enviar pedido al proveedor
Proveedor

5.2. Actividades

5.2.1. Website

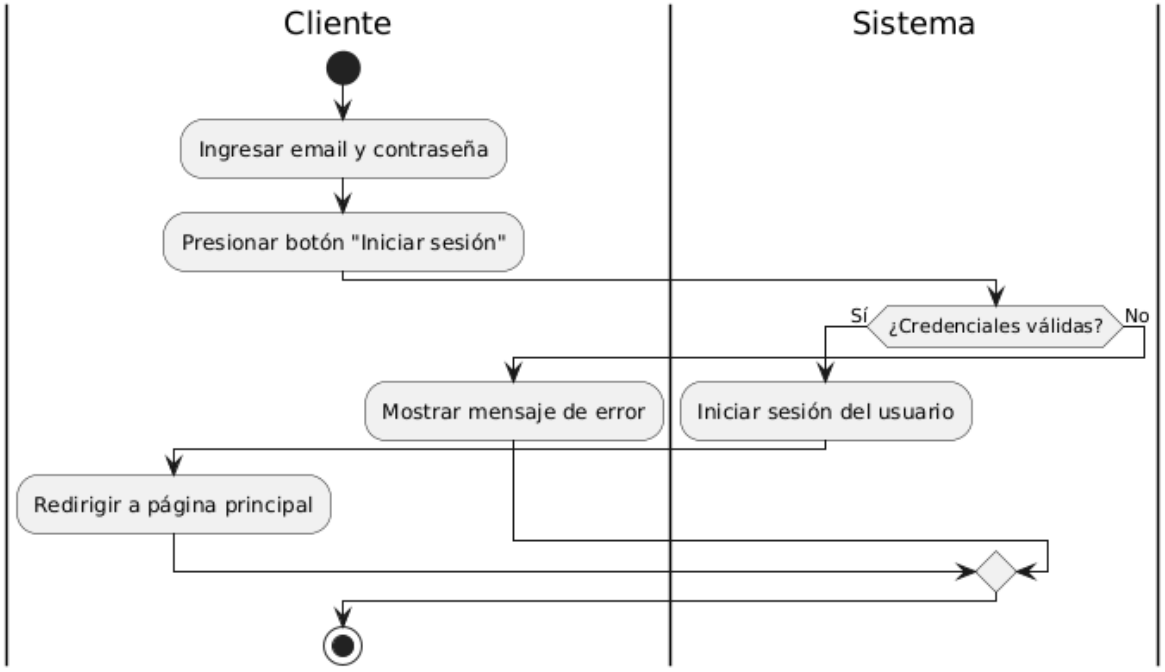


Figura 5.4: Diagrama de Actividad - Iniciar Sesión (Clientes)

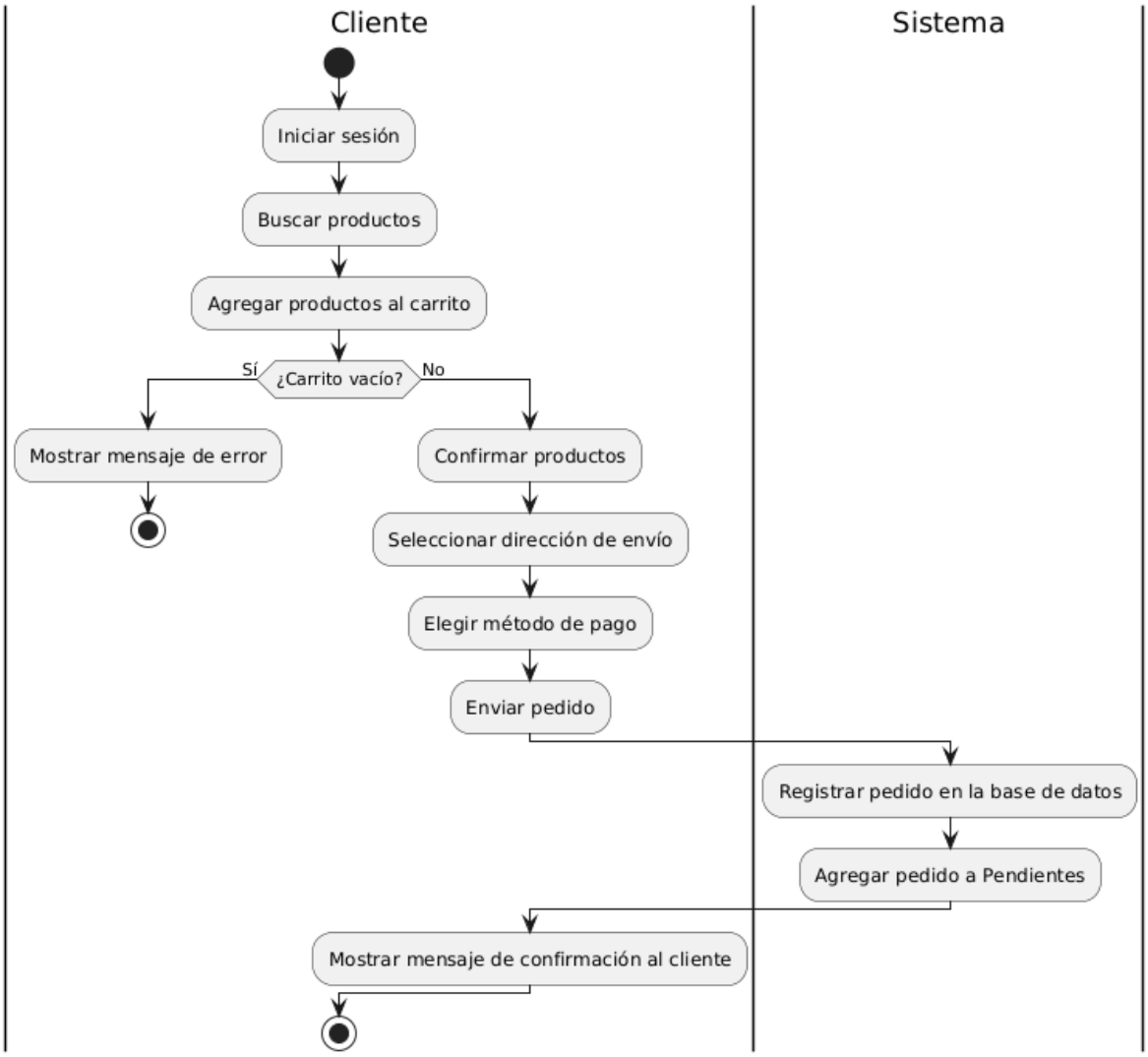


Figura 5.5: Diagrama de Actividad - Realizar Pedido (Clientes)

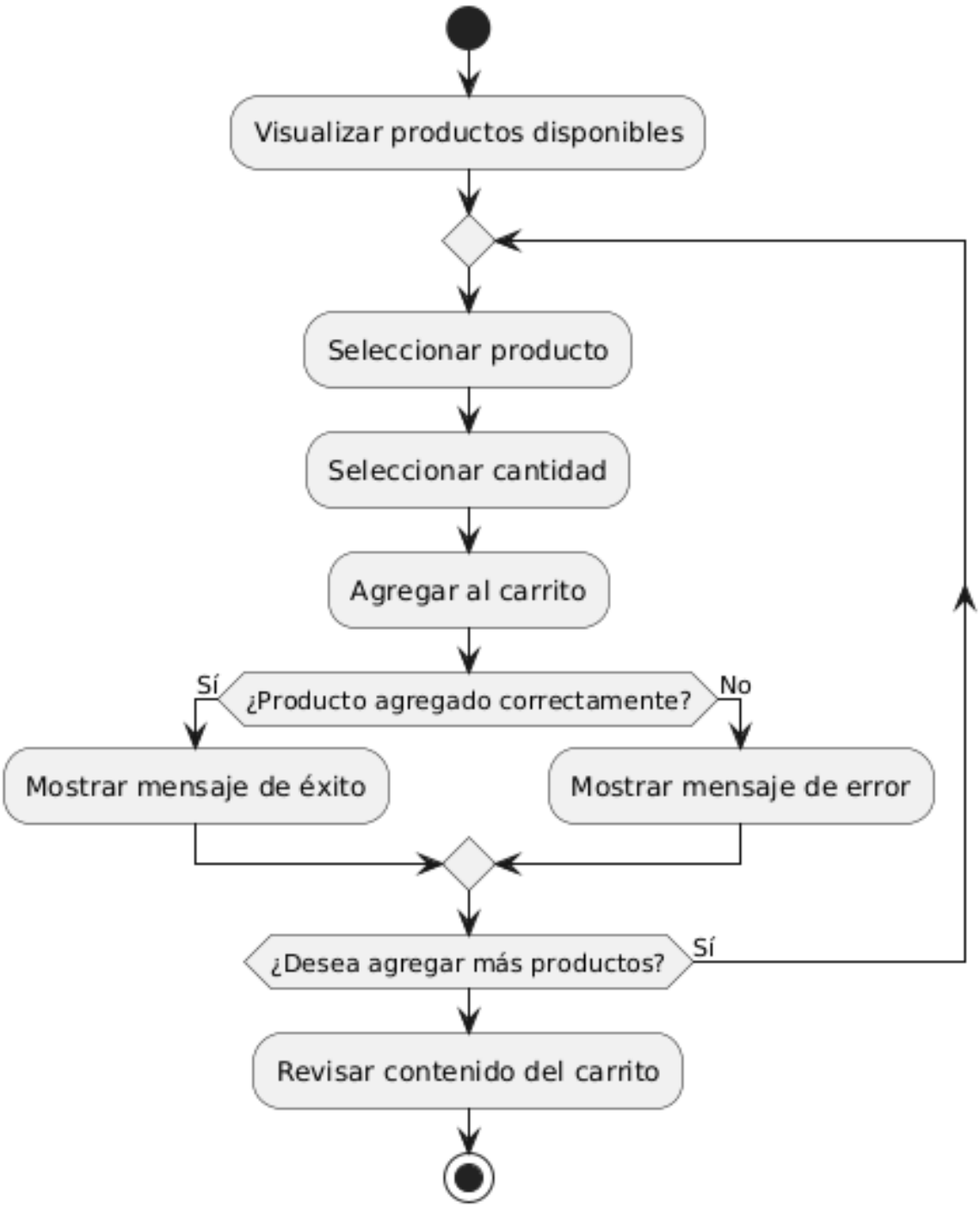


Figura 5.6: Diagrama de Actividad - Agregar al carrito (Clientes)

5.2.2. Admnistrador

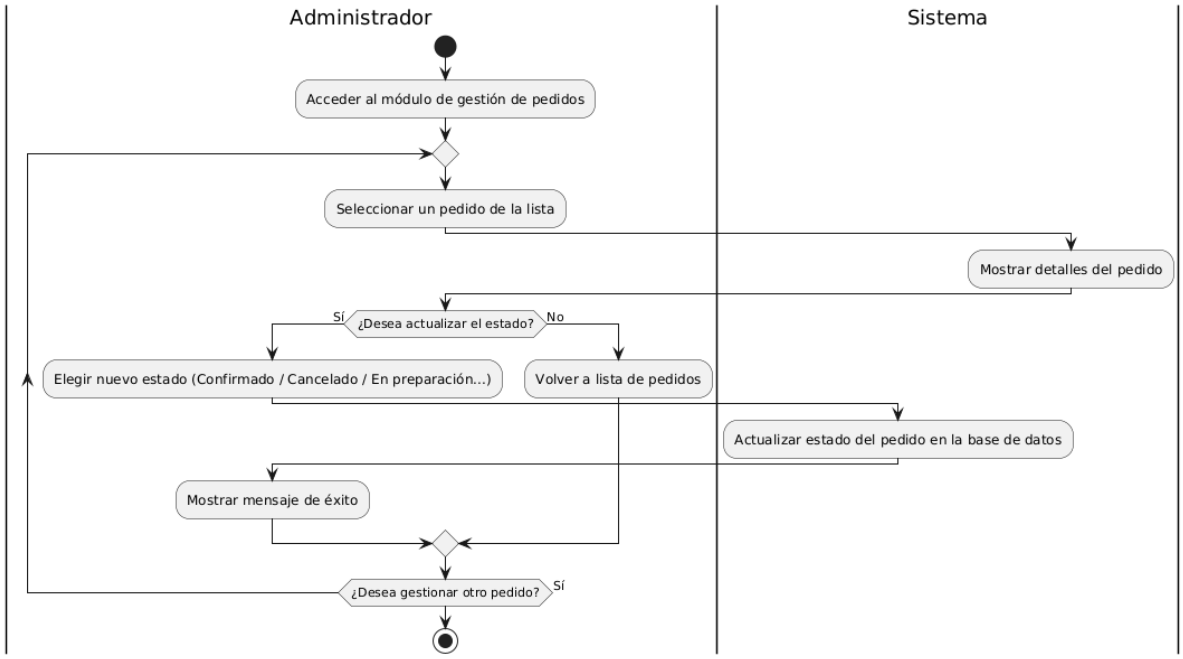


Figura 5.7: Diagrama de Actividad - Manejo de Pedidos de Clientes (Administrador)

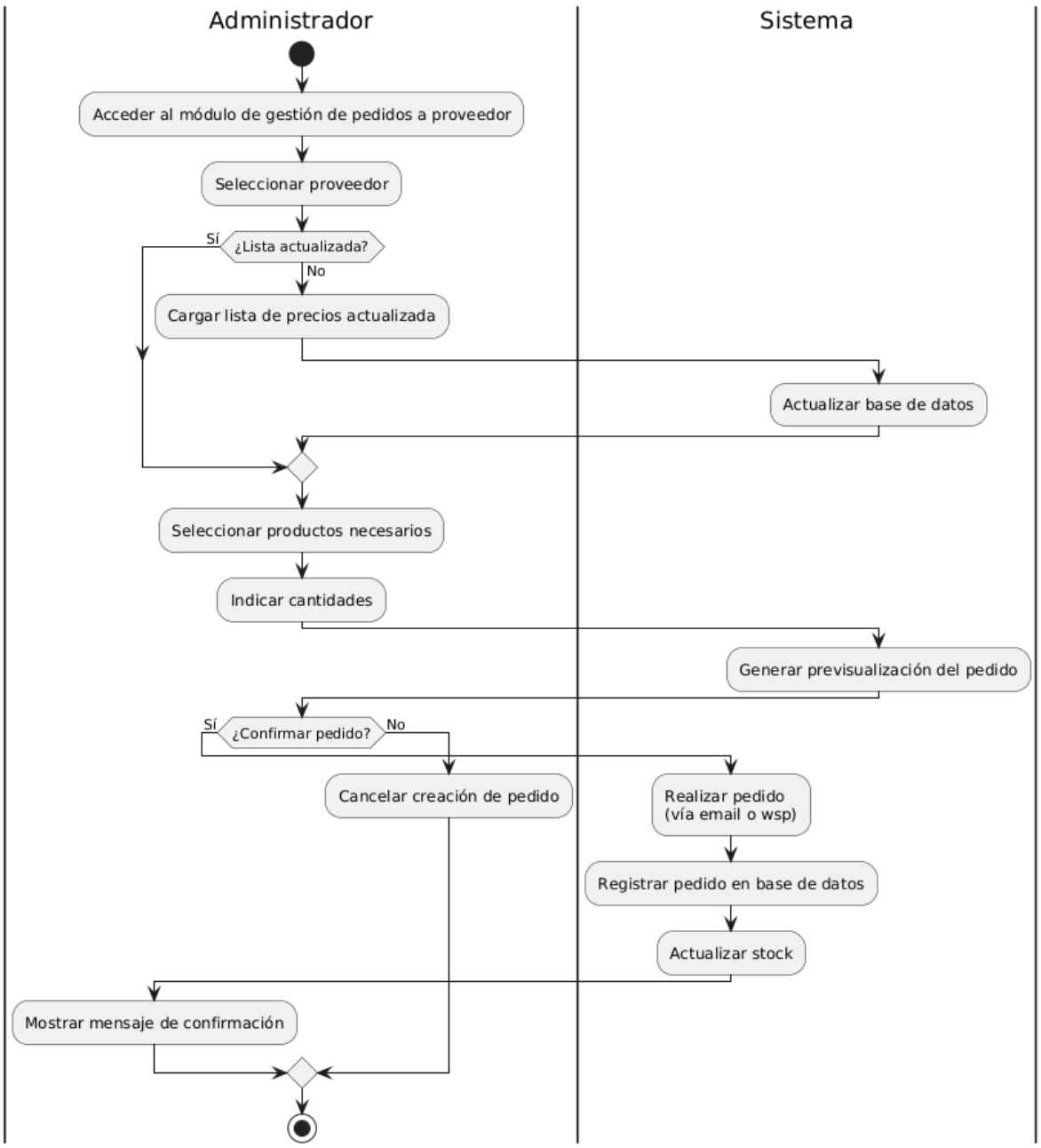


Figura 5.8: Diagrama de Actividad - Manejo de Pedidos a Proveedores (Administrador)

5.2.3. Automatizaciones

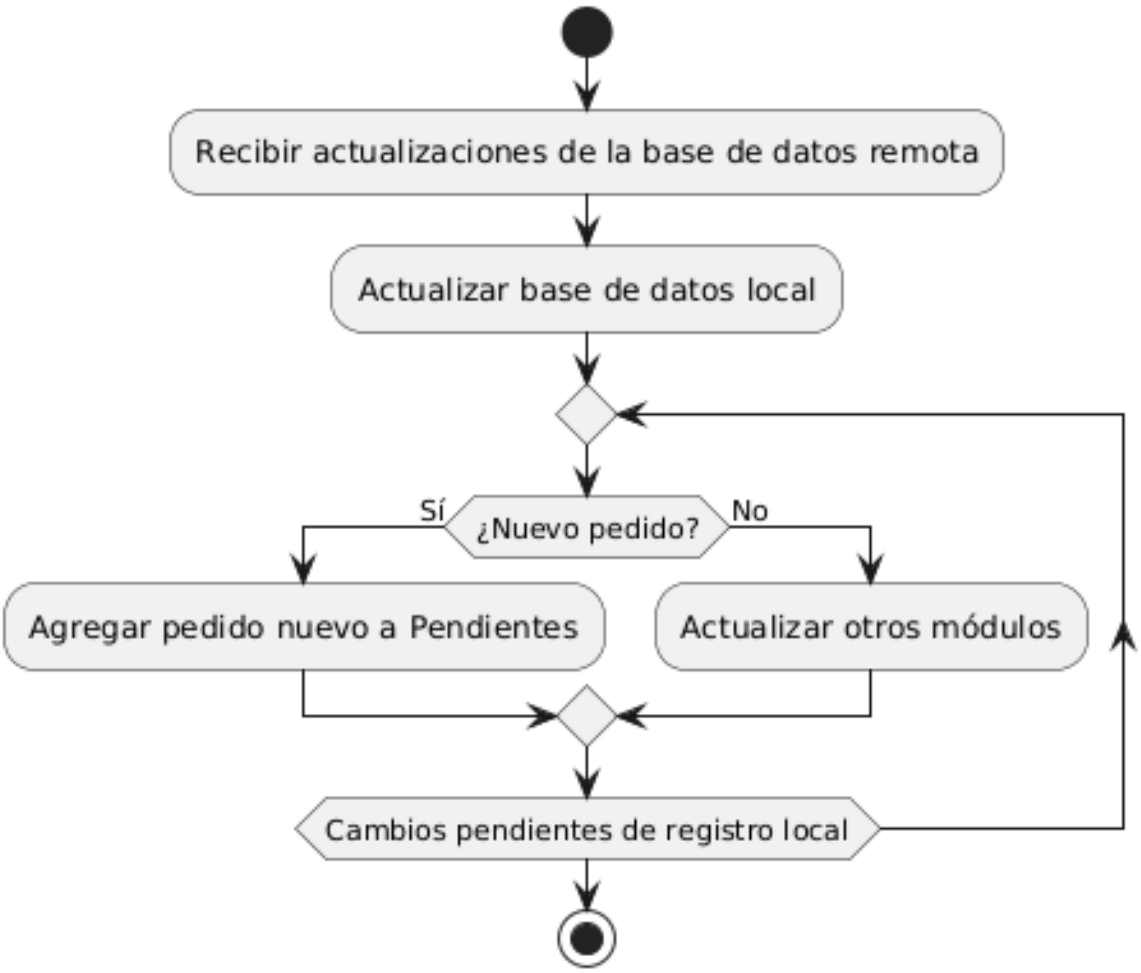


Figura 5.9: Diagrama de Actividad - Actualizar BD (Sistema)

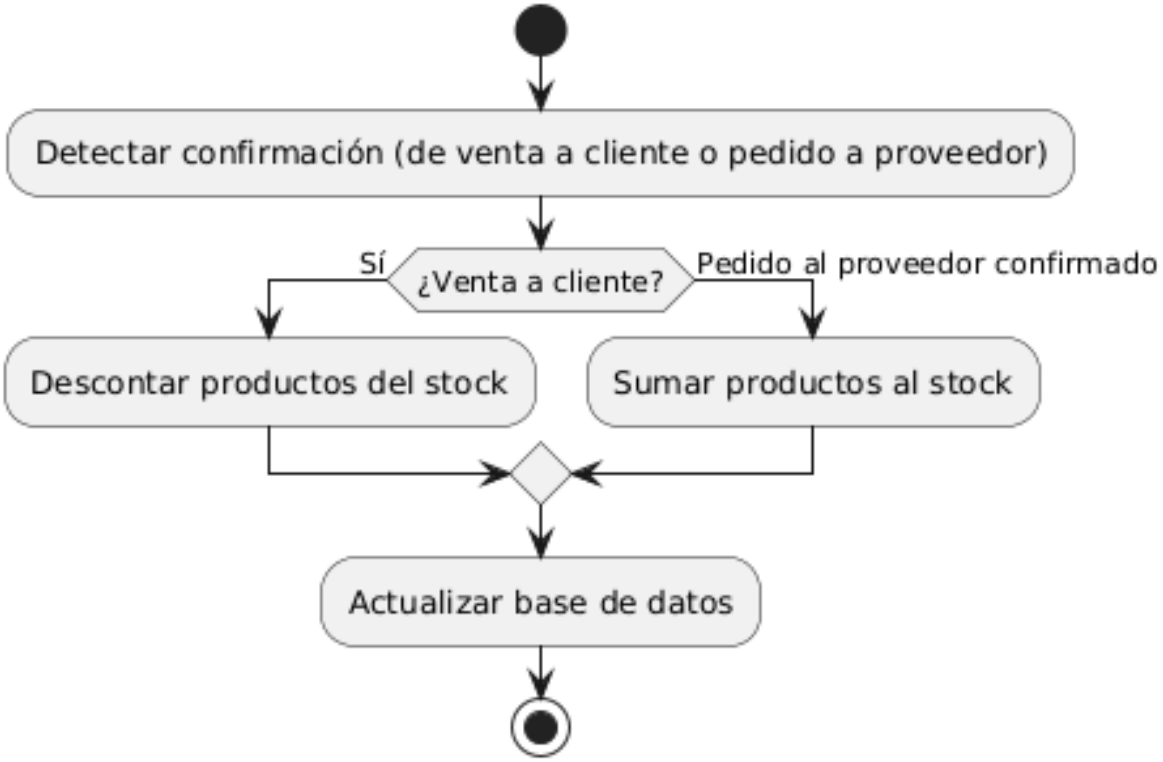


Figura 5.10: Diagrama de Actividad - Actualizar Stock de Productos (Sistema)

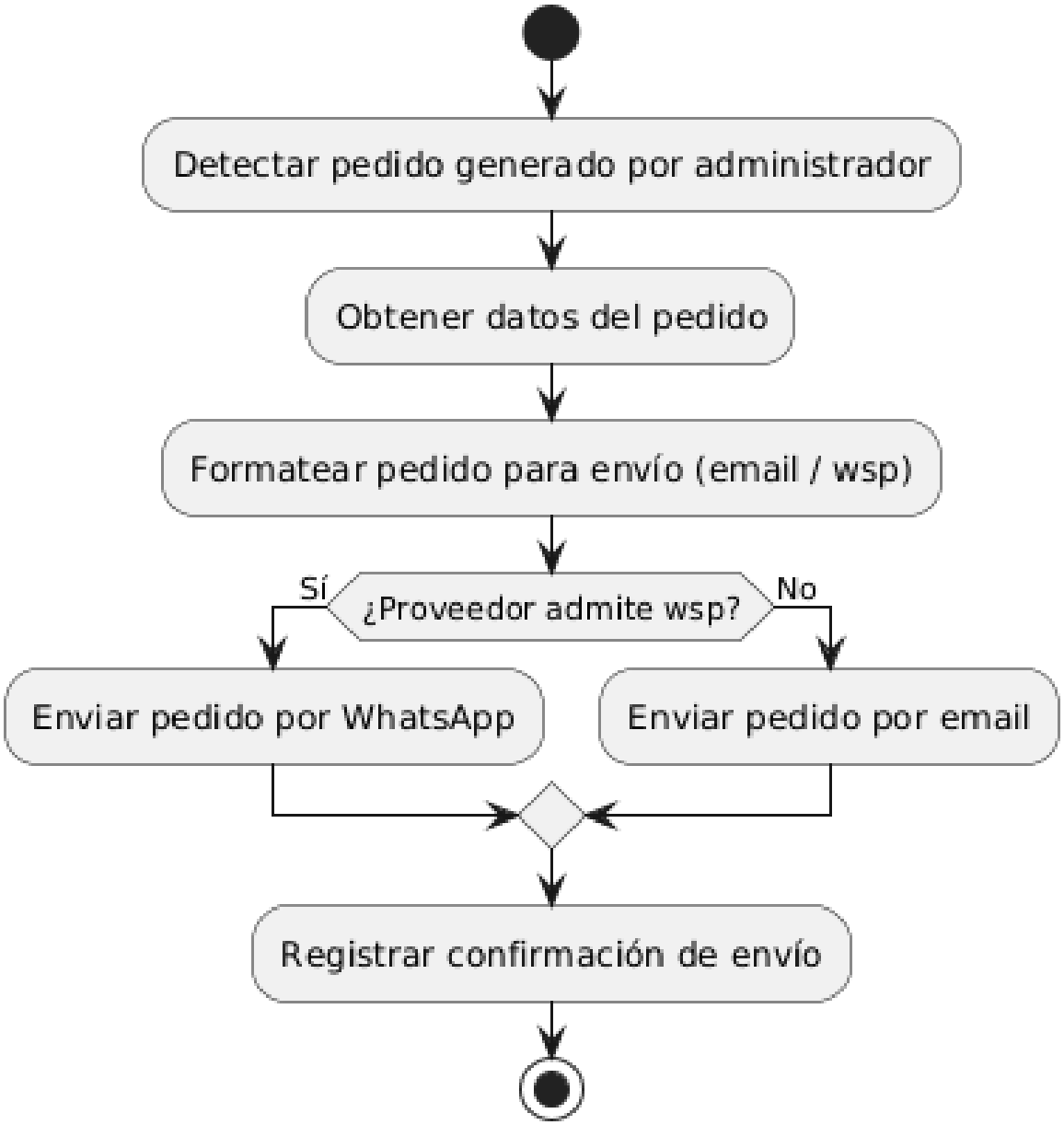


Figura 5.11: Diagrama de Actividad - Automatización envío de pedido a Proveedor (Sistema)

5.3. Secuencia

5.3.1. Plataforma Web

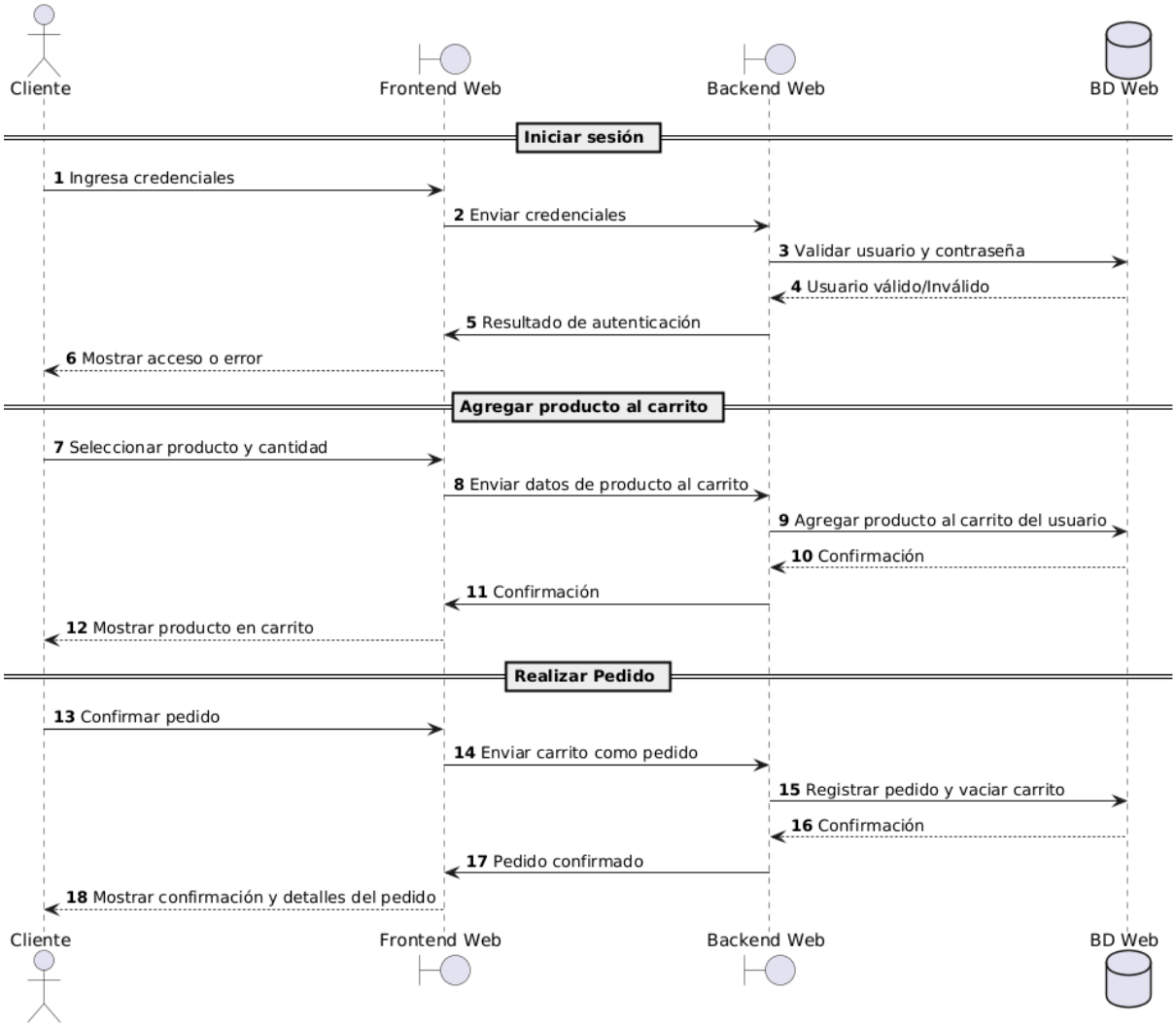


Figura 5.12: Diagrama de Secuencia - Plataforma Web

5.3.2. Administrador

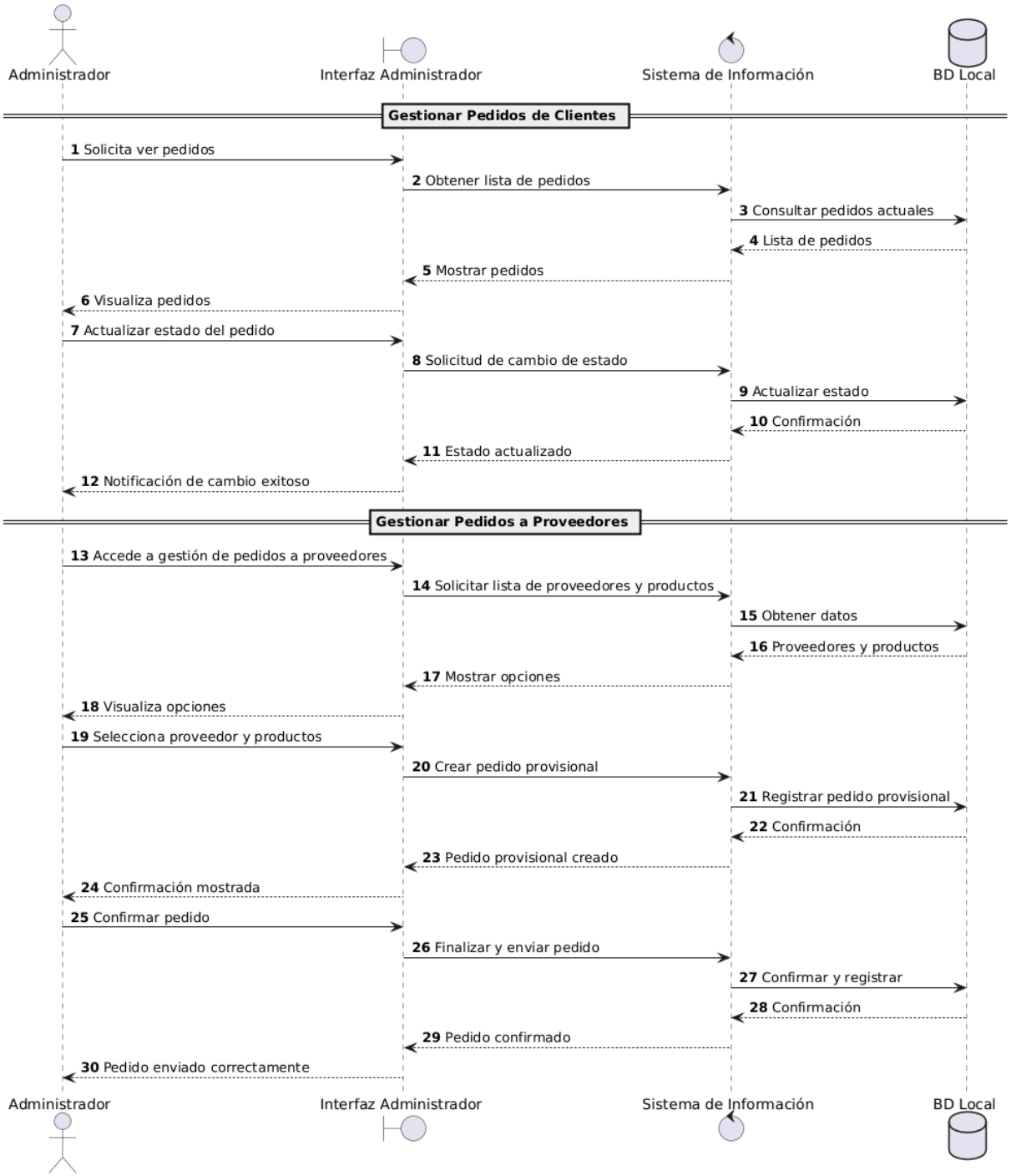


Figura 5.13: Diagrama de Secuencia - Administrador

5.3.3. Automatizaciones

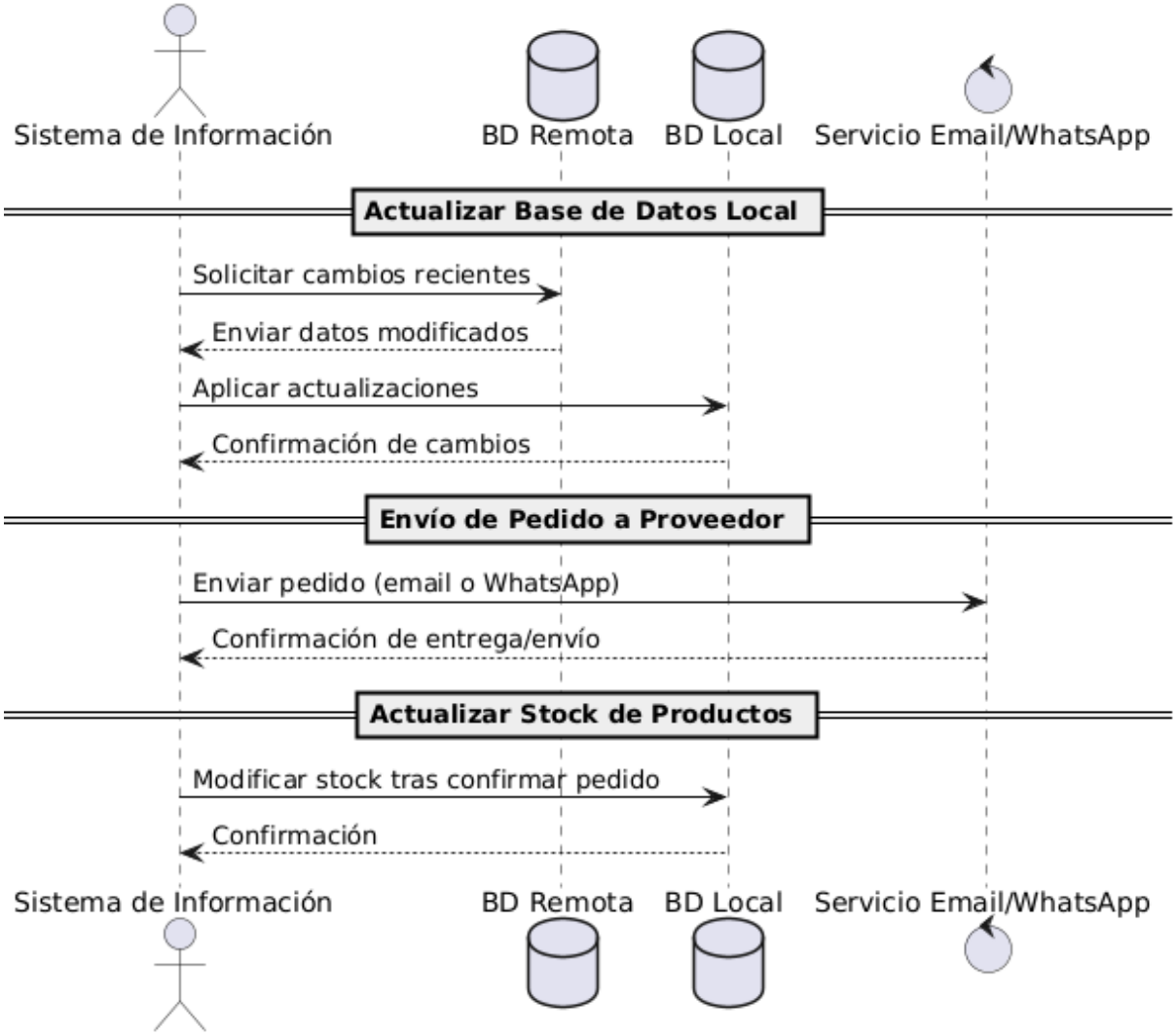


Figura 5.14: Diagrama de Secuencia - Automatizaciones SI Local

5.4. Clases

5.4.1. Tarjetas CRC

Las tarjetas CRC (Clase-Responsabilidad-Colaboración) representan las entidades potenciales del sistema. Y responden a las preguntas (Clase) ¿Quién es?, (Responsabilidad) ¿Qué debe realizar esta clase?, y (Colaboración) ¿Con qué otras entidades interactúa?.

Cuadro 5.1: Tarjeta CRC: Cliente

Cliente	
Responsabilidades	**Colaboraciones**
▪ Registrarse e iniciar sesión en la plataforma. ▪ Consultar productos. ▪ Agregar productos al carrito. ▪ Realizar pedidos.	▪ Plataforma Web ▪ Carrito ▪ Pedido

Cuadro 5.2: Tarjeta CRC: Administrador

Administrador	
Responsabilidades	**Colaboraciones**
▪ Visualizar y gestionar pedidos de clientes. ▪ Confirmar y cancelar pedidos. ▪ Crear y confirmar pedidos a proveedores.	▪ Sistema de Información ▪ Pedido ▪ Producto ▪ Proveedor

Cuadro 5.3: Tarjeta CRC: Sistema de Información

Sistema de Información	
Responsabilidades	**Colaboraciones**
▪ Automatizar la recepción de pedidos de clientes. ▪ Automatizar el envío de pedidos a proveedores. ▪ Actualizar el stock de productos. ▪ Recibir listas de precios de los proveedores.	▪ BD Local ▪ Proveedor ▪ Pedido ▪ Producto

Cuadro 5.4: Tarjeta CRC: Pedido

Pedido	
Responsabilidades	**Colaboraciones**
▪ Representar un pedido de cliente o proveedor. ▪ Mantener el estado del pedido. ▪ Asociar productos y cantidades.	▪ Cliente ▪ Administrador ▪ Producto ▪ Carrito

Cuadro 5.5: Tarjeta CRC: Producto

Producto	
Responsabilidades	**Colaboraciones**
▪ Representar un producto disponible para la venta. ▪ Indicar precio, stock y proveedor asociado.	▪ Pedido ▪ Carrito ▪ Proveedor

Cuadro 5.6: Tarjeta CRC: Proveedor

Proveedor	
Responsabilidades	**Colaboraciones**
▪ Enviar lista de precios al sistema. ▪ Recibir pedidos desde el sistema.	▪ Sistema de Información ▪ Producto ▪ Pedido

5.4.2. Diagramas

Dominio

A continuación se va a desarrollar el Diagrama de Dominio (también conocido como Diagrama de Clases estático de alto nivel) a partir de las tarjetas CRC ya definidas.
Relaciones clave:

- Cliente — (1 a 1) — Carrito

- Cliente — (1 a muchos) — Pedido
- Administrador — (1 a muchos) — Pedido
- Administrador — (1 a muchos) — Producto (gestiona)
- Sistema de Información — automatiza — Pedido y Producto
- Proveedor — (1 a muchos) — Producto
- Proveedor — (1 a muchos) — Pedido
- Pedido — (1 a muchos) — Producto

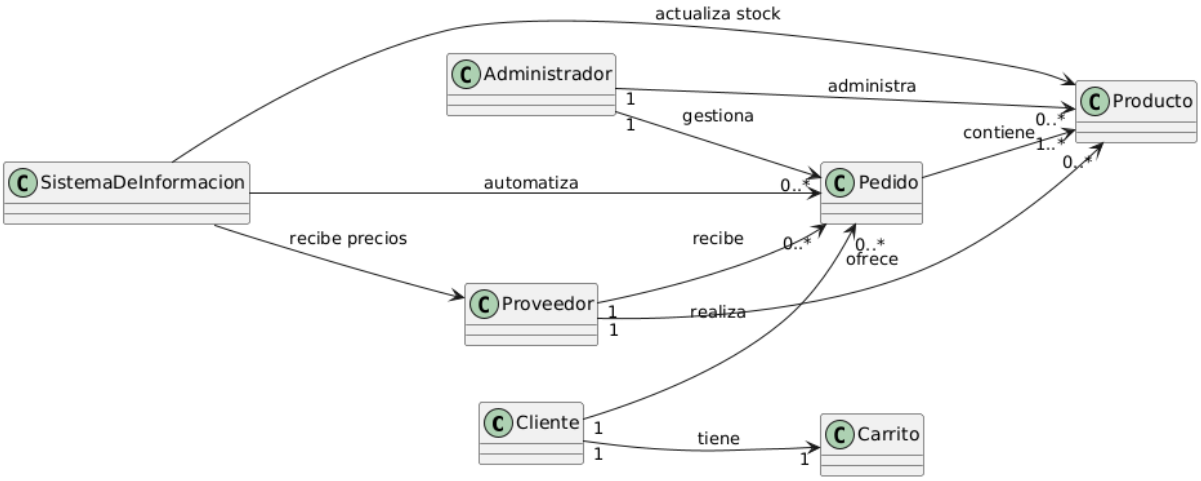


Figura 5.15: Diagrama de Clases - Dominio

General

Seguidamente el diagrama de Clases con sus atributos y métodos correspondientes.

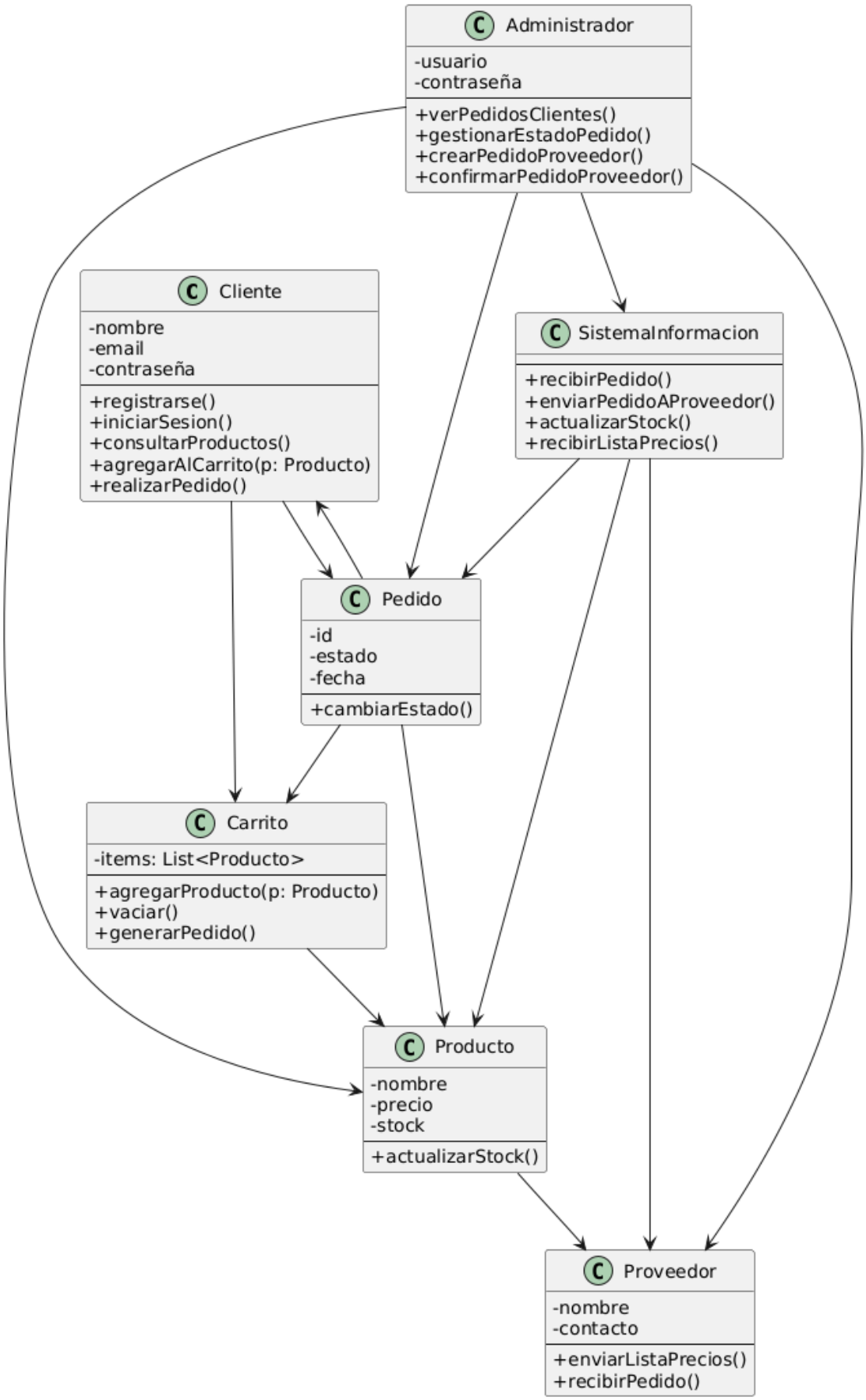


Figura 5.16: Diagrama de Clases - General

5.5. Colaboración

5.5.1. Colaboración entre clases

Los diagramas de comunicación (también conocidos como diagramas de colaboración) permiten visualizar las interacciones entre objetos en el sistema, destacando cómo cooperan entre sí para realizar una funcionalidad específica. A diferencia de los diagramas de secuencia, se enfocan más en la estructura y menos en el tiempo, representando los mensajes numerados que viajan entre instancias de clases.

A continuación, se presentan los diagramas de comunicación para los escenarios a desarrollar del sistema.

5.5.2. Plataforma web

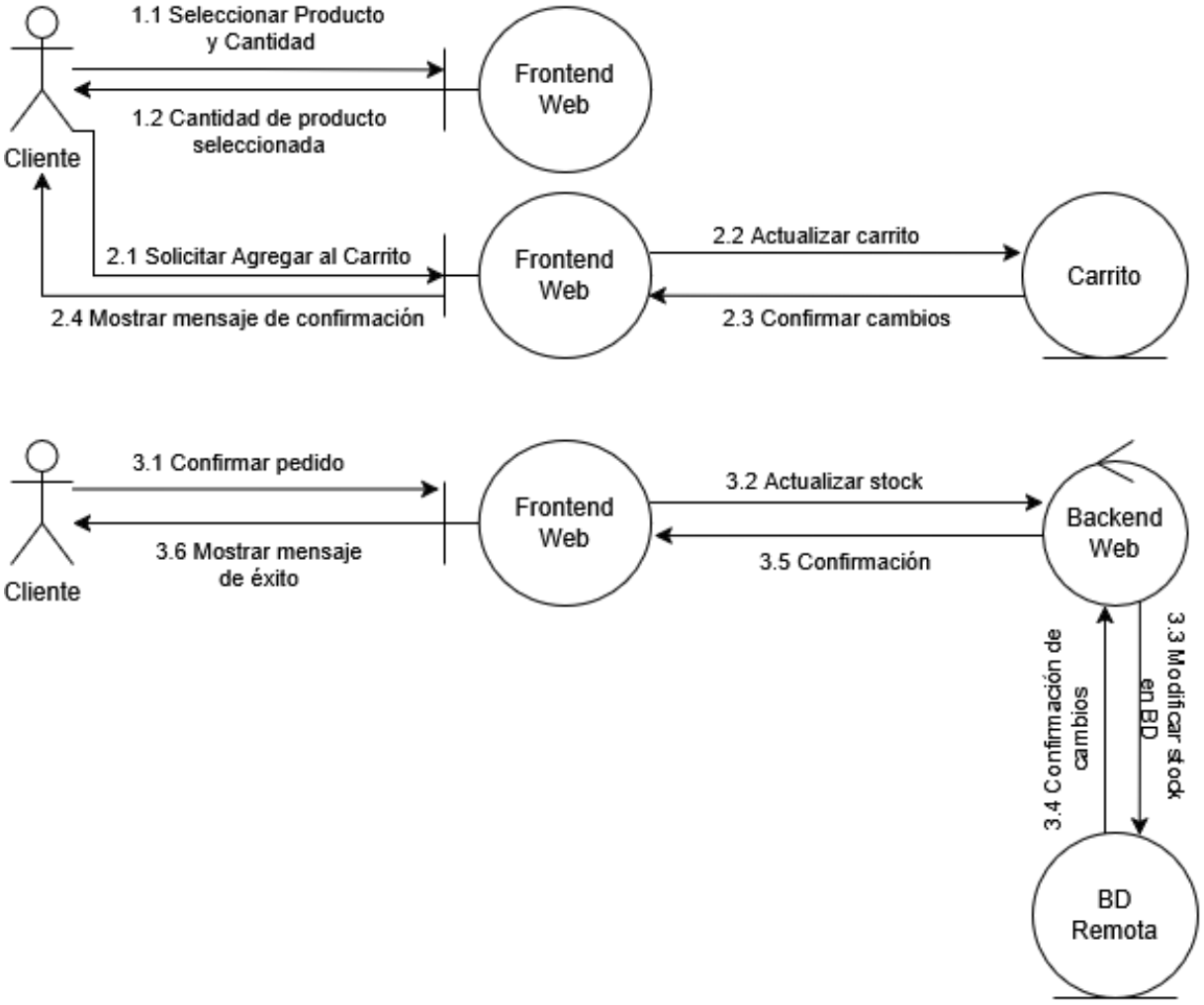


Figura 5.17: Diagrama de colaboración - Plataforma Web

5.5.3. Sistema de Información Local

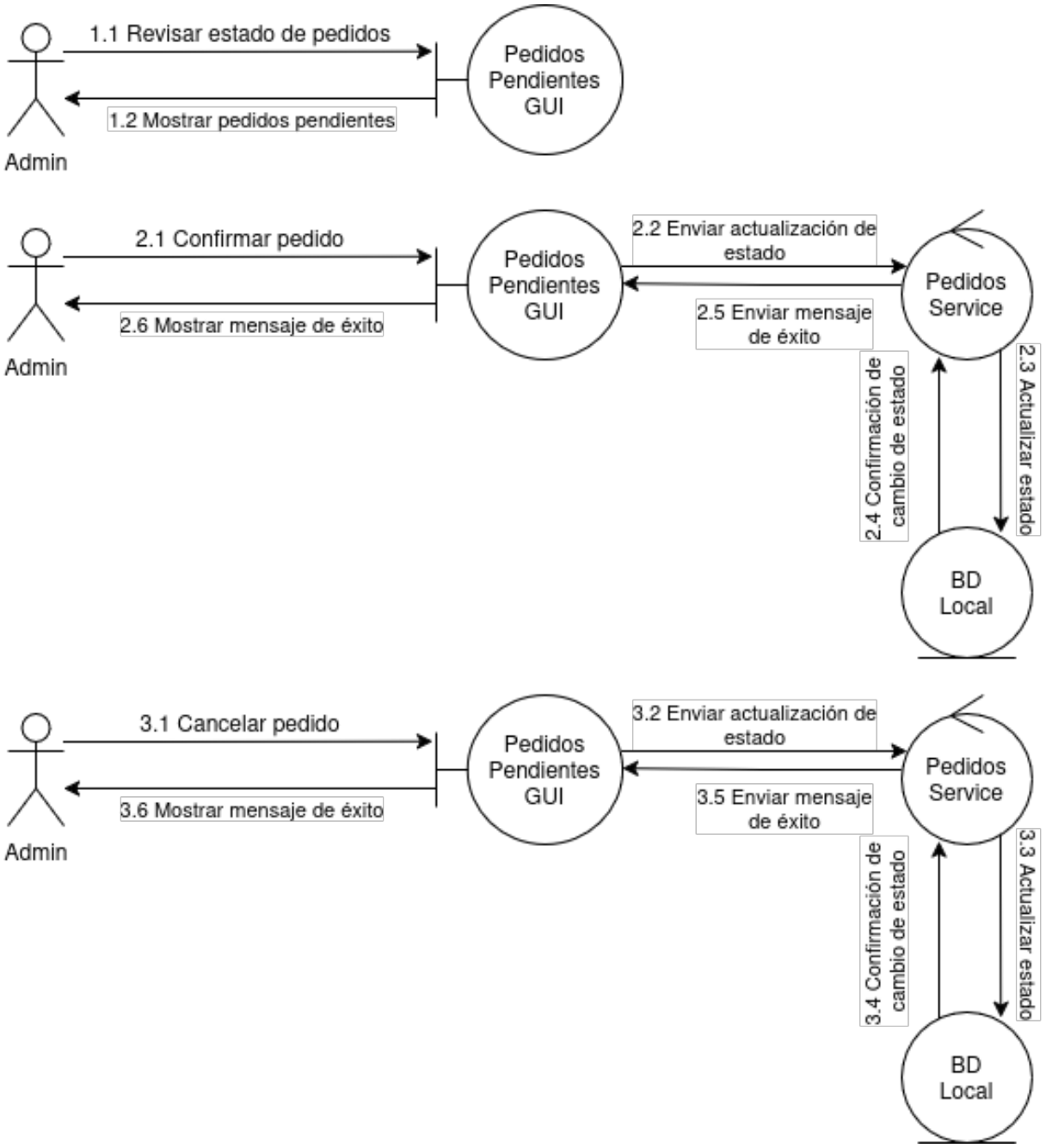


Figura 5.18: Diagrama de colaboración - Sistema de Información Local

Bibliografía

[1] Kent et al. El manifiesto Ágil, 2001. Disponible en `https://agilemanifesto.org/`.

[2] Introducción a uml. Disponible en `https://www.uml.org/what-is-uml.htm`.

[3] Uml diagram types (with examples). Disponible en `https://creately.com/blog/diagrams/uml-diagram-types-examples/`.