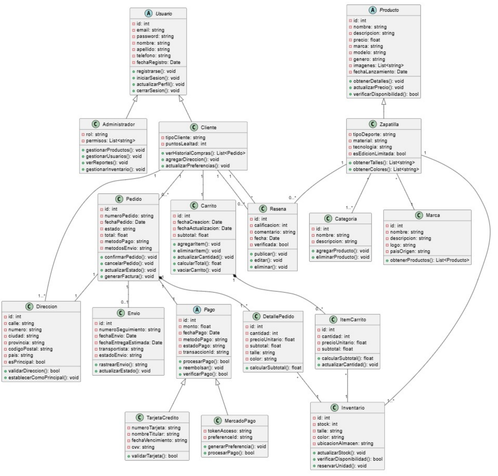
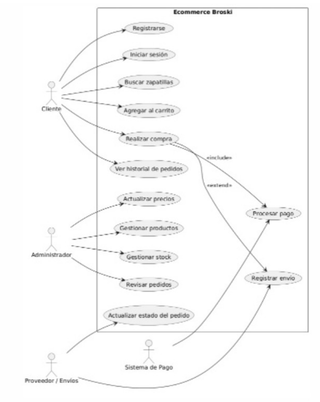
**Explicación del Diagrama de Clases**

- **Producto**: representa los artículos disponibles para compra.
- **Carrito**: clase centrada en la lógica del carrito, manipulando la sesión del usuario.
- **ItemCarrito**: representa cada producto dentro del carrito, con su cantidad y subtotal.
- **Orden**: entidad que representa una compra finalizada.

La relación principal muestra que un carrito contiene múltiples items, y cada item está vinculado a un producto.



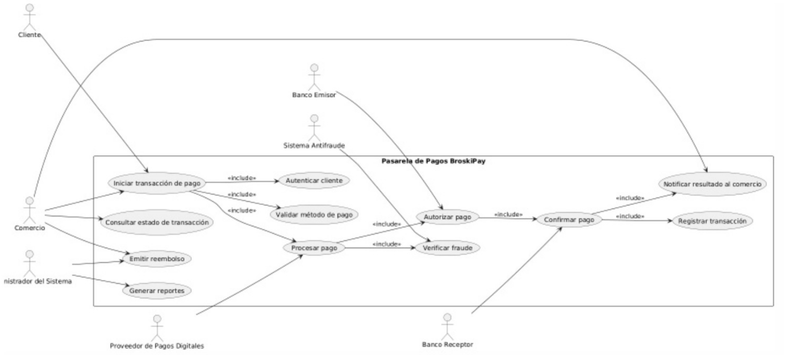

**Explicación del Diagrama de Caso de Uso**

El diagrama identifica al **Usuario** como el actor principal que interactúa con el sistema.

Los casos de uso representan las funciones esenciales:

- **Navegar Productos**: permite ver el catálogo.
- **Ver Detalle de Producto**: despliega información específica de un ítem.
- **Agregar Producto al Carrito**: incorpora un producto a la sesión del usuario.
- **Actualizar Cantidades**: modifica unidades seleccionadas.
- **Eliminar Producto**: remueve un ítem de la lista.
- **Visualizar Carrito**: muestra el contenido actual.
- **Realizar Compra**: proceso final que simula o confirma la orden.

Este conjunto de casos de uso cubre el flujo típico de un sistema de compras básico.




**Decisiones de Diseño**

**✓ Uso de Django y patrón MVT**

Se eligió Django por su arquitectura modular basada en el patrón **Model–View–Template**, lo que permite un desarrollo ágil, mantenible y escalable.

**✓ Carrito basado en sesión**

El carrito se implementó mediante la sesión del usuario para evitar dependencia inmediata de la base de datos, logrando:

- mejor rendimiento
- simplicidad inicial
- independencia del inicio de sesión

**✓ Separación en apps: productos y carrito**

Esta decisión aumenta la cohesión y reduce el acoplamiento, permitiendo reutilizar o extender cada módulo de forma independiente.

**✓ Modelo Producto simple**

Se optó por un modelo de datos básico para productos, suficiente para un sistema de ejemplo y escalable a futuro.

**✓ Lógica del carrito encapsulada en una clase**

Permite:

- reutilización en múltiples vistas
- consistencia en cálculos y reglas
- mantenimiento centralizado

**✓ Uso opcional de Bootstrap**

Permite una interfaz responsiva sin agregar complejidad al backend.