



Objetivo del diagrama de clases

La creación de este diagrama tiene la intención de guiar y estructurar el modelo de POO que será utilizado para crear una página web para la tienda de ropa Merced. En él se presentan diferentes clases que se alinean con el modelo de datos (Diagrama de Entidad Relación).

El sistema gestiona usuarios, productos, carritos, pedidos, reseñas y pagos; asegurando consistencia entre datos y operaciones.

Usuario

Rol

Representa a la persona que navega, compra y/o administra la tienda. Sus relaciones son primordiales, posee un carrito activo, genera pedidos y puede crear reseñas de los productos.

Atributos

- id: int – clave primaria.
- nombre, apellido– datos personales del usuario.
- mail: str – único.
- contraseña: str – dato de acceso del usuario, deberá tener un mínimo de 6 caracteres, 1 letra y 1 número.
- rol: str– estándar / admin.
- dni: str – único

Métodos

- es_admin(): bool – verifica si el usuario tiene rol de administrador.
- validar_login(mail, contraseña): bool – comprueba si las credenciales coinciden con alguna registrada.

Relaciones

- 1 — 1 Carrito (único carrito activo).
- 1 — * Pedido histórico(historial de todos los pedidos).
- 1 — * Dirección. Un usuario puede poseer varias direcciones.
- 1 — * Resena. Un usuario puede escribir múltiples reseñas en diferentes productos.

Gestor_usuario

Rol

Controla la gestión de datos de los usuarios. Permite crear, modificar, buscar y eliminar (CRUD).

Atributos

- usuarios: list[Usuario] – lista que contiene todas las instancias de Usuario.

Métodos

- registrar_usuario(nombre, apellido, mail, contrasena, dni): recibe todos los datos necesarios para crear un usuario con su respectiva clase.
- login_usuario(mail, contrasena): comprueba si las credenciales existen y son correctas, devuelve el usuario.
- buscar_usuario(id): devuelve datos de un usuario específico basado en su id.
- listar_usuarios(): devuelve un listado de todos los usuarios registrados.
- eliminar_usuario(id): elimina a un usuario por su id.
- modificar_datos(id, dato_opcion, dato_valor): permite actualizar algún dato del alumno a la vez (nombre, apellido, contrasena), recibe su id para localizar el cambio, el número de la opción elegida que son equivalentes a cada dato que se puede modificar y el nuevo valor del dato.
- modificar_rol(id, rol): cambia el rol de un usuario.

Relaciones

- Usa/crea/gestiona Usuario.
- Es usado por Menu

Producto

Rol

Modelo de un artículo vendible, presenta sus datos tanto para el usuario final como para el vendedor (stock).

Atributos

- id_producto: int (Primary Key)
- nombre: str
- descripción: str
- talla: str
- marca: str
- categoria: str (por ejemplo: zapatos, remera, etc.)
- color: str
- precio: decimal
- imagenes: list[str] – URLs/rutas.
- stock: int

Métodos

- actualizar_stock(cantidad): void
- cambiar_precio(precio): void

Relaciones

- 1 — * resena
- 1 — * Detalle_carrito / 1 — * Detalle_pedido
- * — 1 Gestor_producto

Gestor_producto

Rol

Capa de negocio para el CRUD del catálogo y búsquedas.

Atributos

- productos: list[Producto] lista de todas las instancias de la clase producto.

Métodos

- agregar_producto(producto): void
- eliminar_producto(id_producto): void
- modificar_producto(id_producto, datos): bool
- buscar_producto(nombre): list[Producto] utilizado en la barra de búsqueda para encontrar productos que coincidan con lo buscado.
- listar_productos(): list[Producto]

Relaciones

- Opera sobre Producto.
- Es utilizado por Menu

Carrito

Rol

Contenedor temporal y editable de la intención de compra del usuario. Solo se tendrá uno y este irá actualizando los precios hasta que prosiga con la siguiente fase: pedido. Modela el estado previo a la compra. No debe contaminar el histórico.

Atributos

- id_carrito: int (Primary Key)
- id_usuario: int (Foreign Key, {Unique})
- items: list[Detalle_carrito]

Métodos

- agregar_producto(producto, cantidad): bool
- eliminar_producto(id_producto): bool
- calcular_total(): decimal
- vaciar_carrito(): void
- actualizar_cantidad(id_producto, cantidad): bool

Relaciones

- 1 Usuario — 1 Carrito
- 1 Carrito — * Detalle_carrito

Detalle_carrito

Rol

Línea del carrito: une un producto con cantidad y calcula el subtotal con el precio vigente. Permite manejar cantidades y recalcular importes antes de confirmar.

Atributos

- id_carrito: int (Foreign Key)
- id_producto: int (Foreign Key)
- precio_unitario: decimal
- cantidad: int

Clave: (id_carrito, id_producto) {PK compuesta}

Método

- subtotal(): decimal— es igual a cantidad * precio_unitario

Relaciones

- 1 Carrito — * Detalle_carrito
- 1 Producto — * Detalle_carrito

Pedido

Rol

Compra confirmada; es histórico e inmutable (salvo estado). Es la prueba de venta: debe guardar precios y cantidades tal como se pagaron.

Atributos

- id_pedido: int (Primary Key)
- id_usuario:int (Foreign Key)
- items: list[Detalle_pedido]
- usuario: Usuario
- direccion_envio: Dirección
- fecha: date/datetime
- total: decimal
- estado: enum limita las opciones a {PENDIENTE, PAGADO, ENVIADO, ENTREGADO, CANCELADO} va a depender del estado de pago.

Métodos

- cambiar_estado(nuevo_estado): void
- calcular_total(): decimal– suma de subtotales(envío+productos).

Relaciones

- 1 Usuario — * Pedido
- 1 Pedido — * Detalle_pedido
- 1 Pedido — 1 Dirección
- 1 Pedido — 1 Pago

Detalle_pedido

Rol

Línea inmutable del pedido: mantiene los precios en el momento de la compra. Permite **trazar** lo pagado, aún si cambian los precios o se actualiza el producto. Cuando se confirma el carrito copia sus elementos actuales.

Atributos

- id_pedido: int (Foreign Key)
- id_producto: int (Foreign Key)
- cantidad: int
- precio_unitario: decimal – congelado al momento de la compra.

Clave: (id_pedido, id_producto) {PK compuesta}

Método

- subtotal(): decimal = cantidad × precio_unitario

Relaciones

- 1 Pedido — * Detalle_pedido
- 1 Producto — * Detalle_pedido

Direccion

Rol

Es el lugar de entrega. Es necesaria para el envío del pedido, un usuario puede guardar más de una dirección.

Atributos

- id_direccion: int (Primary Key)
- id_usuario: int (Foreign Key)
- calle, ciudad, provincia, codigo_postal, pais, telefono: str

Relaciones

- 1 Usuario — * Direccion
- 1 Pedido — 1 Direccion

Pago

Rol

Información de cobro del pedido. Permite conciliar el cobro y proseguir o detener la operación del pedido.

Atributos

- id_pago: int (Primary Key)
- metodo: enum { TARJETA, TRANSFERENCIA, PAYPAL, ...}.
- monto: decimal
- estado: enum {PENDIENTE, APROBADO, RECHAZADO} una vez alterado modifica el estado del pedido.
- fecha: date/datetime

Relaciones

- 1 Pedido — 1 Pago

Resena

Rol

Es el feedback del cliente sobre un producto. Aumenta la confianza de los clientes; además fomenta la retroalimentación del catálogo.

Atributos

- id_resena: int (Primary Key)
- id_usuario: int (Foreign Key)
- id_producto: int (Foreign Key)
- texto: str
- estrellas: int (1..5)
- imagenes: list[str] (atributo opcional)
- fecha: date/datetime

Métodos

- escribir_resena(usuario, texto, estrellas): void
- eliminar_resena(id_resena): bool

Relaciones

- 1 Usuario — * resena
- 1 Producto — * resena

Validador

Rol

Realiza validaciones de input para verificar que se reciben datos con el formato esperado y si no están ya utilizados en otras instancias.

Métodos

- validar_opcion(min, max): verifica que la opción ingresada esté dentro del rango válido.
- validar_mail_patron(mail): comprueba si el formato del correo es válido.

- validar_mail_repetido(mail): comprueba si el correo ya se encuentra en uso.
- validar_contrasena(contrasena): válida que la contrasena tenga un mínimo de 6 caracteres, 1 letra y 1 número.
- validar_dni(dni): verifica el formato y longitud del DNI, así mismo comprueba que no esté en uso.
- validar_str(cadena): asegura que el campo no esté vacío ni contenga caracteres inválidos.

Menu

Rol

Orquesta flujos de menú apoyándose en los gestores. Actúa como interfaz entre el usuario y el sistema.

Atributos

- usuario_actual: Usuario
- gestor: Gestor_usuario
- validador: Validador
- gestor_producto: Gestor_producto

Métodos

- mostrar_menu(opciones): int
- menu_principal() / menu_admin() / menu_estandar() / menu_productos(): void

Flujo del negocio

Checkout, flujo de compra (Carrito → Pedido)

1. Usuario llena su Carrito.
2. Al confirmar → se crea un Pedido (estado = pendiente) con:
3. Detalle_pedido generado de los Detalle_carrito.
4. Dirección seleccionada del usuario.

5. Se crea Pago (estado = pendiente).
6. Si pago aceptado → Pedido.estado = confirmado.
Si rechazado → Pedido.estado = cancelado.
7. Si el pago es aprobado → descontar stock (en Producto o VarianteProducto).
8. El Carrito vuelve a quedar vacío.

Notas importantes

- En base de datos, los precios y montos se almacenan como decimales.
- Se evita el uso de las Ñ y las tildes en las clases para evitar posibles problemas/errores.
-

Restricciones de integridad

- No permitir stock negativo.
- cantidad ≥ 1 .
- Totales consistentes con subtotales.
- El mail y dni deben ser valores únicos.