ABP TIENDA VIRTUAL

ENUNCIADO:

La tienda virtual que se hará es de todo tipo de **calzado**, en el que se dispondrá a la venta estos mismos.

Parte Administrador:

Una interfaz sencilla donde puedas ver los productos y categorías que existen en la tienda, de éstos, se podrán editar, eliminar (desactivar) y añadir más. También, se podrá consultar las comandas que se han hecho y filtrarlas según el usuario que las ha hecho, o el estado en el que se encuentran. Si una comanda está en estado pendiente, el administrador podrá cambiar el estado a enviado y colocar, automáticamente, la fecha de entrega.

Parte comprador (sin validar):

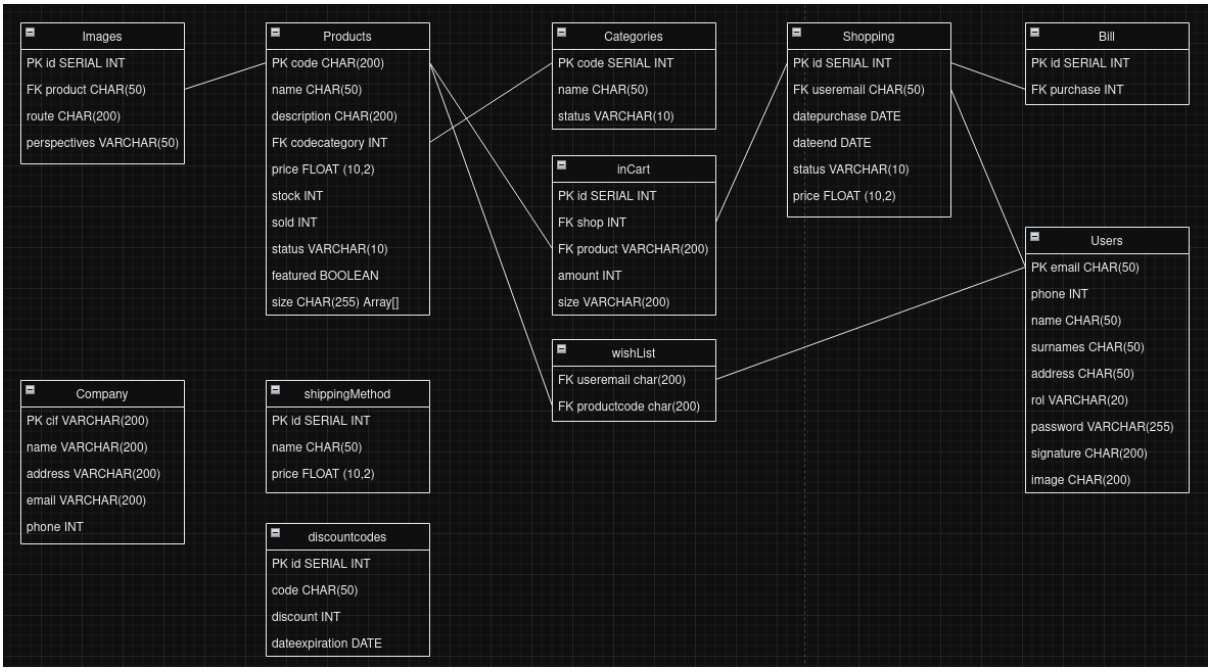
Una interfaz donde se puede consultar todos los productos existentes disponibles, podrá incluso añadir a un carrito los productos, el cual, no se guardará en la próxima sesión, a no ser de que este inicie la sesión.

Parte comprador (validado):

Una interfaz similar a la del comprador sin validar, con la diferencia de que este puede modificar su perfil, puede añadir productos al carrito y que el carrito se guarde en la base de datos, también, podrá realizar compras de estos carritos, y generar facturas una vez “realizadas”.

MODELO RELACIONAL:

products(**code**, name, description, *codecategory*, price, stock, size, sold, status, featured)
wishList(***useremail***, ***productcode***)
images(**id**, perspectives, *product*, route)
categories(**code**, name, status)
users(**email**, password, phone, name, surnames, address, rol, signature, image)
inCart(**id,** *product*, *shop*, amount, size)
shopping(**id,** *useremail*, price, status, datepurchase, dateend)
bill(**id**, *purchase*)
shippingMethod(**id**, name, price)
discountCodes(**id**, code, discount, dateexpiration)



MODEL E-R:

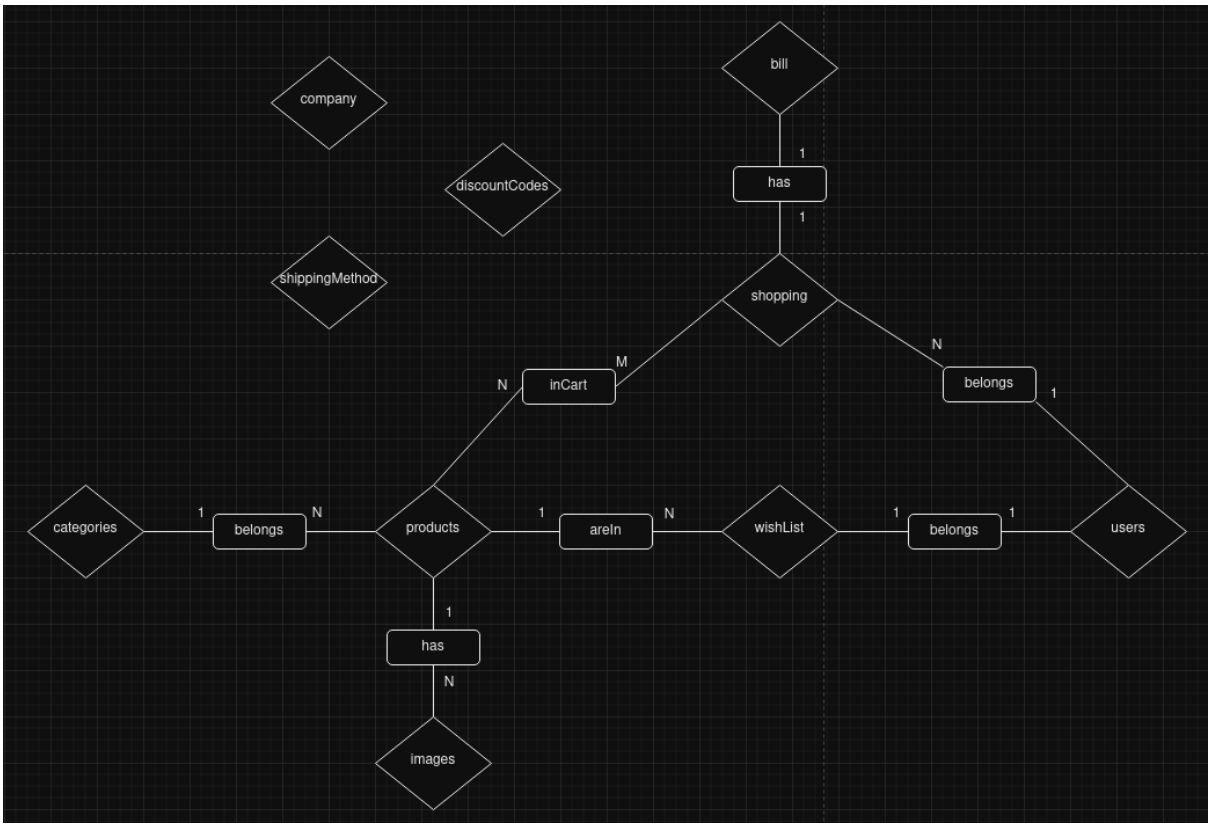

bill
company
1
has
discountCodes
1
shippingMethod
shopping
N
M
N
inCart
belongs
1
categories
1
belongs
N
products
1
areIn
N
wishList
1
belongs
1
users
1
has
N
images

DIAGRAMA DE CLASES:

Creación de la base de datos (PostgreSQL):

BDD:

```
postgres=# CREATE DATABASE urbanstore
postgres-# ;
CREATE DATABASE
postgres=# \c urbanstore
You are now connected to database "urbanstore" as user "postgres".
urbanstore=#
```

Tabla "*categories*":

```
urbanstore=# CREATE TABLE categories (
urbanstore(#     code CHAR(50) PRIMARY KEY,
urbanstore(#     name CHAR(50) NOT NULL
urbanstore(# );
CREATE TABLE
urbanstore=#
```

Tabla "*products*":

```
urbanstore=# CREATE TABLE products (
urbanstore(#     code CHAR(50) PRIMARY KEY,
urbanstore(#     name CHAR(50) NOT NULL,
urbanstore(#     description CHAR(200),
urbanstore(#     codeCategory CHAR(50) REFERENCES categories(code) ON UPDATE CASCADE ON DELETE CASCADE,
urbanstore(#     price INT NOT NULL,
urbanstore(#     stock INT NOT NULL,
urbanstore(#     outstanding BOOLEAN NOT NULL
urbanstore(# );
CREATE TABLE
```

Alguns canvis:

```
urbanstore=# ALTER TABLE products
urbanstore-# ADD COLUMN size INT NOT NULL;
ALTER TABLE
```

```
urbanstore=# ALTER TABLE products
urbanstore-# ALTER COLUMN outstanding TYPE CHAR(50);
ALTER TABLE
```

```
urbanstore=# ALTER TABLE products
urbanstore-# ADD CONSTRAINT fk_products_outstanding
urbanstore-#     FOREIGN KEY (outstanding)
urbanstore-#     REFERENCES wishList(code)
urbanstore-#     ON UPDATE CASCADE
urbanstore-#     ON DELETE CASCADE;
ALTER TABLE
urbanstore=#
```

```
urbanstore=# ALTER TABLE products
urbanstore-# ADD COLUMN sold INT NOT NULL;
ALTER TABLE
urbanstore=#
```

Tabla “*Images”*:

```
urbanstore=# CREATE TABLE images (
urbanstore(#    id CHAR(50) PRIMARY KEY,
urbanstore(#    product CHAR(50) REFERENCES products(code) ON UPDATE CASCADE ON DELETE CASCADE NOT NULL,
urbanstore(#    route CHAR(200) NOT NULL
urbanstore(# );
CREATE TABLE
urbanstore=#
```

Alguns canvis:

```
urbanstore=# ALTER TABLE images
urbanstore-# ADD COLUMN perspectives CHAR(50) NOT NULL;
ALTER TABLE
```

Perspectives Imatges:

lateralperspective

aboveperspective

belowperspective

Tabla “*Users*”:

```
urbanstore=# CREATE TABLE users (
urbanstore(#    email CHAR(50) PRIMARY KEY,
urbanstore(#    phone INT NOT NULL,
urbanstore(#    name CHAR(50) NOT NULL,
urbanstore(#    surnames CHAR(50) NOT NULL,
urbanstore(#    address CHAR(50),
urbanstore(#    rol VARCHAR(20) CHECK (rol IN ('admin', 'customer')) NOT NULL
urbanstore(# );
CREATE TABLE
urbanstore=#
```

Alguns canvis:

ALTER TABLE users
ALTER COLUMN phone DROP NOT NULL;

ALTER TABLE users
ALTER COLUMN address DROP NOT NULL;

ALTER TABLE users
ADD COLUMN image;

Tabla “*Bill*”:

```
urbanstore=# CREATE TABLE bill (
urbanstore(#    id CHAR(50) PRIMARY KEY,
urbanstore(#    purchase CHAR(50) REFERENCES shopping(id) ON UPDATE CASCADE ON DELETE CASCADE NOT NULL,
urbanstore(#    userEmail CHAR(50) REFERENCES users(email) ON UPDATE CASCADE ON DELETE CASCADE NOT NULL
urbanstore(# );
CREATE TABLE
urbanstore=#
```

Alguns canvis:

```
urbanstore=# ALTER TABLE bill
urbanstore-# DROP COLUMN userEmail;
ALTER TABLE
urbanstore=#
```

Tabla “*notifications*”:

```
urbanstore=# CREATE TABLE notifications (
urbanstore(#    id CHAR(50) PRIMARY KEY,
urbanstore(#    userEmail CHAR(50) REFERENCES users(email) ON UPDATE CASCADE ON DELETE CASCADE NOT NULL,
urbanstore(#    message CHAR(50) NOT NULL,
urbanstore(#    title CHAR(50) NOT NULL
urbanstore(# );
CREATE TABLE
urbanstore=#
```

Tabla “*wishList*”:

```
urbanstore=# CREATE TABLE wishList (
urbanstore(#    userEmail CHAR(255) REFERENCES users(email) ON DELETE CASCADE ON UPDATE CASCADE NOT NULL,
urbanstore(#    productCode INT REFERENCES products(code) ON DELETE CASCADE ON UPDATE CASCADE NOT NULL,
urbanstore(#    PRIMARY KEY (useremail, productcode)
urbanstore(# );
CREATE TABLE
urbanstore=#
```

Tabla “*shopping*”:

```
urbanstore=# CREATE TABLE shopping (
urbanstore(#    id CHAR(50) PRIMARY KEY,
urbanstore(#    userEmail CHAR(50) REFERENCES users(email) ON UPDATE CASCADE ON DELETE CASCADE NOT NULL,
urbanstore(#    price INT,
urbanstore(#    status VARCHAR(20) CHECK (status IN ('pendiente', 'enviado')) NOT NULL,
urbanstore(#    datePurchase DATE,
urbanstore(#    dateEnd DATE
urbanstore(# );
CREATE TABLE
urbanstore=#
```

Alguns canvis:

```
urbanstore=# ALTER TABLE shopping
urbanstore-# DROP CONSTRAINT IF EXISTS shopping_status_check;
ALTER TABLE
urbanstore=# ALTER TABLE shopping
urbanstore-# ADD CONSTRAINT check_shopping_status
urbanstore-#    CHECK (status IN ('pending', 'shipped', 'cart'));
ALTER TABLE
urbanstore=#
```

Tabla "*inCart*":

```
urbanstore=# CREATE TABLE inCart (
urbanstore(#    id CHAR(50) PRIMARY KEY,
urbanstore(#    product CHAR(50) REFERENCES products(code) ON UPDATE CASCADE ON DELETE CASCADE NO
T NULL
urbanstore(#    shop CHAR(50) REFERENCES shopping(id) ON UPDATE CASCADE ON DELETE CASCADE NOT NUL
L
```

```
urbanstore(# );
CREATE TABLE
urbanstore=#
```

Alguns canvis:

```
urbanstore=# ALTER TABLE inCart
urbanstore-# ADD COLUMN amount CHAR(50) NOT NULL
urbanstore-# ;
ALTER TABLE
urbanstore=#
```

ALTER TABLE inCart
ADD COLUMN price int NOT NULL;

Exportar la base de dades:

Necessari ficar-se a la carpeta de PostgreSQL, en la carpeta bin, i fer la següent comanda:

pg_dump -U postgres ***urbanstore*** *> Ruta_Fins_A_Escriptori\urbanstore.sql*

Importar la base de dades:

Necessari crear abans la base de dades: *CREATE DATABASE urbanstore;* Executar la següent comanda:

psql -U postgres ***urbanstore*** *< Ruta_Fins_A_Arxiu_SQL*

Conexión PostgreSQL-PHP

Tenemos que entrar en nuestra carpeta ‘xampp’, dentro de ella entraremos en ‘php’ y abriremos el archivo php.ini, buscaremos las siguientes líneas y eliminaremos el ‘;’.

```
extension=pdo_pgsql
extension=pdo_sqlite
extension=pgsql
```

Database.php:

```
<?php

class Database{
	protected static $conn;

	public function __construct() {
	self::$conn = self::connect();
	}
	public static function connect() {

	try {
	// Configuración de la base de datos
	$servername = "localhost";
	$username = "root";
	$password = "";
	$dbname = "urbanstore";

	// Crear conexión
	$connectPdo = new PDO('pgsql:host='.$servername.';dbname='.$dbname, $username, $password);
	$connectPdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
	// Establecer el conjunto de caracteres a UTF-8
	$connectPdo->exec("SET NAMES 'utf8'");
	return $connectPdo;

	} catch (PDOException $e) {
	echo "Error de PDO: " . $e->getMessage();
	}

	}

}

?>
```

Nuevo archivo Database.php con la función de “getConnection”:

```
<?php

class Database{
        protected static $conn;
        public static function connect() {

        try {
        // Configuración de la base de datos
        $servername = "localhost";
        $username = "postgres";
        $password = "password";
        $dbname = "urbanstore";

        // Crear conexión
        $connectPdo = new PDO('pgsql:host='.$servername.';dbname='.$dbname,
$username, $password);
        $connectPdo->setAttribute(PDO::ATTR_ERRMODE,
PDO::ERRMODE_EXCEPTION);
        // Establecer el conjunto de caracteres a UTF-8
        $connectPdo->exec("SET NAMES 'utf8'");
        return $connectPdo;

        } catch (PDOException $e) {
        echo "Error de PDO: " . $e->getMessage();
        }

        }

        protected static function getConnection() {
      // Verificar si ya hay una conexión existente
      if (!isset(self::$conn) || self::$conn === null) {
            // Si no hay una conexión, crear una nueva instancia de Database y establecer la
conexión
        $database = new self();
        self::$conn = $database->connect();
      }

      // Devolver la conexión existente
      return self::$conn;
   }

}

?>
```

CREACIÓN PULIDA DE LA BASE DE DATOS:

BDD:

```
postgres=# CREATE DATABASE urbanstore;
CREATE DATABASE
postgres=# \c urbanstore;
You are now connected to database "urbanstore" as user "postgres".
urbanstore=#
```

Categories:

```
urbanstore=# CREATE TABLE categories (
urbanstore(#    code SERIAL PRIMARY KEY,
urbanstore(#    name CHAR(50) NOT NULL
urbanstore(# );
CREATE TABLE
urbanstore=#
```

```
urbanstore=# ALTER TABLE CATEGORIES
urbanstore-# ADD COLUMN status VARCHAR(10) CHECK (status IN ('enabled','disabled'));
ALTER TABLE
```

```
urbanstore=# ALTER TABLE categories
urbanstore-# ALTER COLUMN status SET NOT NULL;
ALTER TABLE
urbanstore=#
```

Products:

El campo “size” ahora es de tipo **array**, puede ser *null*.

```
urbanstore=# DROP TABLE products;
DROP TABLE
urbanstore=# CREATE TABLE products (
urbanstore(#    code CHAR(200) PRIMARY KEY,
urbanstore(#    name CHAR(50) NOT NULL,
urbanstore(#    description CHAR(200) NOT NULL,
urbanstore(#    codecategory INT REFERENCES categories(code) ON UPDATE CASCADE ON DELETE CASCADE NOT NULL,
urbanstore(#    price NUMERIC(10, 2) NOT NULL,
urbanstore(#    stock int NOT NULL,
urbanstore(#    size int NOT NULL,
urbanstore(#    sold int NOT NULL,
urbanstore(#    status VARCHAR(10) NOT NULL CHECK (status IN ('enabled', 'disabled'))
urbanstore(# );
CREATE TABLE
urbanstore=#
```

```
urbanstore=# ALTER TABLE products
urbanstore-# ADD COLUMN featured BOOLEAN;
ALTER TABLE
```

Images:

```
urbanstore=# CREATE TABLE images (
urbanstore(#    id SERIAL PRIMARY KEY,
urbanstore(#    product CHAR(50) REFERENCES products(code) ON UPDATE CASCADE ON DELETE CASCADE NOT NULL,
urbanstore(#    route CHAR(200) NOT NULL,
urbanstore(#    perspectives VARCHAR(50) NOT NULL CHECK (perspectives IN ('lateralperspective','aboveperspective','belowperspective','3dmodel'))
urbanstore(# );
CREATE TABLE
urbanstore=#
```

Users:

```
urbanstore=# CREATE TABLE users (
urbanstore(#    email CHAR(50) PRIMARY KEY,
urbanstore(#    phone INT,
urbanstore(#    name CHAR(50) NOT NULL,
urbanstore(#    surnames CHAR(50) NOT NULL,
urbanstore(#    address CHAR(50),
urbanstore(#    rol VARCHAR(20) NOT NULL CHECK (rol IN ('admin', 'customer')),
urbanstore(#    password CHAR(200) NOT NULL,
urbanstore(#    signature CHAR(200),
urbanstore(#    image CHAR(200)
urbanstore(# );
CREATE TABLE
urbanstore=#
```

```
urbanstore=# ALTER TABLE users
urbanstore-# ALTER COLUMN password TYPE CHAR(255);
ALTER TABLE
urbanstore=#
```

Notifications:

```
urbanstore=# CREATE TABLE notifications (
urbanstore(#    id SERIAL PRIMARY KEY,
urbanstore(#    useremail CHAR(50) REFERENCES users(email) ON UPDATE CASCADE ON DELETE CASCADE NOT NULL,
urbanstore(#    message CHAR(50) NOT NULL,
urbanstore(#    title CHAR(50) NOT NULL
urbanstore(# );
CREATE TABLE
urbanstore=#
```

wishList:

```
urbanstore=# CREATE TABLE wishList (
urbanstore(#    userEmail CHAR(255) REFERENCES users(email) ON DELETE CASCADE ON UPDATE CASCADE NOT NULL,
urbanstore(#    productCode CHAR(255) REFERENCES products(code) ON DELETE CASCADE ON UPDATE CASCADE NOT NULL,
urbanstore(#    PRIMARY KEY (userEmail, productCode)
urbanstore(# );
CREATE TABLE
urbanstore=#
```

Shopping:

```
urbanstore=# CREATE TABLE shopping (
urbanstore(#    id SERIAL PRIMARY KEY,
urbanstore(#    userEmail char(50) REFERENCES users(email) ON DELETE CASCADE ON UPDATE CASCADE NOT NULL,
urbanstore(#    price INT,
urbanstore(#    status VARCHAR(20) NOT NULL CHECK (status IN ('enabled','disabled')),
urbanstore(#    datePurchase DATE,
urbanstore(#    dateEnd DATE
urbanstore(# );
CREATE TABLE
urbanstore=#
```

```
urbanstore=# ALTER TABLE shopping
urbanstore-# ADD COLUMN status VARCHAR(10) CHECK (status IN ('cart','shipped','pending'));
ALTER TABLE
urbanstore=#
```

InCart:

```
urbanstore=# CREATE TABLE inCart (
urbanstore(# id SERIAL PRIMARY KEY,
urbanstore(# shop INT NOT NULL,
urbanstore(# product VARCHAR(200),
urbanstore(# amount INT NOT NULL,
urbanstore(# size VARCHAR(200)
urbanstore(# );
CREATE TABLE
urbanstore=#
```

Bill:

```
urbanstore=# CREATE TABLE bill (
urbanstore(#    id SERIAL PRIMARY KEY,
urbanstore(#    purchase INT REFERENCES shopping(id) ON UPDATE CASCADE ON DELETE CASCADE NOT NULL
urbanstore(# );
CREATE TABLE
urbanstore=#
```

shippingMethod:

```
urbanstore=# CREATE TABLE shippingMethod (
urbanstore(#    id SERIAL PRIMARY KEY,
urbanstore(#    name CHAR(50) NOT NULL,
urbanstore(#    price NUMERIC(10, 2) NOT NULL
urbanstore(# );
CREATE TABLE
```

discountCodes:

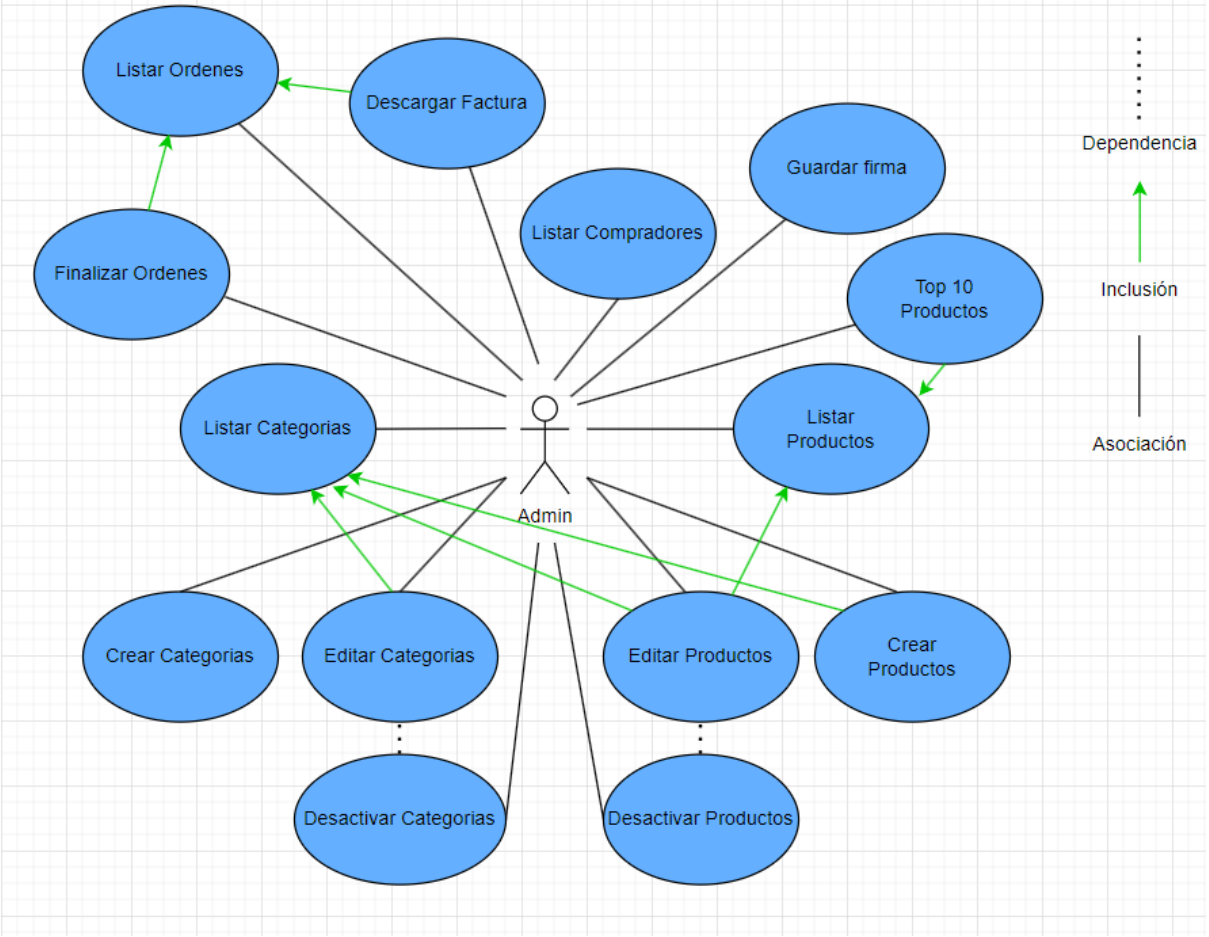
```
urbanstore=# CREATE TABLE discountCodes (
urbanstore(#    id SERIAL PRIMARY KEY,
urbanstore(#    code CHAR(50) NOT NULL,
urbanstore(#    discount INT NOT NULL,
urbanstore(#    dateexpiration DATE NOT NULL
urbanstore(# );
CREATE TABLE
urbanstore=#
```

Codigos de descuento predeterminados:

UrbanStore : 10%
croissant: 15%
AF1: 10%
running: 18%
paplo: 33%

Estos codigos caducan todos en 2025, menos croissant, que estara caducado.

DIAGRAMA DE CASOS DE USO ADMIN:



Guardar firma:

1. El administrador solicita guardar su firma dando al botón de modificar la firma.
2. El sistema le devolverá un canvas.
3. El administrador dibuja su firma en el canvas y le da al botón de aceptar.
4. El sistema recoge el dibujo y lo guarda como la firma del administrador, si ya hay una firma existente, se reemplaza por la nueva.

Listar productos:

1. El administrador llega al apartado de productos.
2. El sistema, automáticamente, le devuelve una lista de productos para mostrar.

Flujo alternativo: Punto 2:

1. El sistema no encuentra productos para mostrar y muestra un mensaje que dice que no hay productos, invitando a añadir nuevos.

Top 10 productos:

1. El administrador está en el *Dashboard* de admin.
2. El sistema, automáticamente, le devuelve una lista de productos para mostrar, pero solo los 10 primeros productos, ordenados por su número de ventas.

Flujo alternativo: Punto 2:

2. El sistema no encuentra productos para mostrar y muestra un mensaje que dice que no hay productos, invitando a añadir nuevos en la pestaña de productos.

Crear productos:

1. El administrador da al botón de crear un nuevo producto.
2. El sistema devuelve un formulario para agregar un producto, con los parámetros de su nombre, descripción, categoría, precio, stock, tamaños, estado y 4 campos para insertar fotos y el modelo 3d, estos estan protegidos para que no se pueda subir ningun archivo con una extensión errónea. La categoría aparecerá en una lista desplegable con su nombre, es decir, primero hay que listar las categorías y los tamaños tendrá un botón para ir añadiendo tantos como quieras.
3. El administrador llena todos los campos y le da al botón de aceptar.
4. El sistema coje los datos, y creará un código único en el cual poderse referir exclusivamente al producto que acaba de crear, después, añadirá finalmente el producto a la base de datos, el campo de ventas se establecerá en 0, al ser un producto nuevo.

Flujo alternativo: Punto 2:

1. El sistema no encuentra categorías, por lo tanto, no podrás crear el producto, ya que es obligatorio que este pertenezca a una categoría, así que no te dejará crear el producto hasta que no hayas creado una categoría para poder asignarla al mismo.

Flujo alternativo: Punto 4:

1. El sistema detecta que cualquiera de los campos no se ha introducido correctamente, este mostrará un mensaje de error, indicando de volver a rellenar los campos correctamente, ningún producto se añadirá.

Editar productos:

1. El administrador da al botón de editar un producto, en un producto en específico de todos los que estan listados.
2. El sistema devuelve un formulario con los campos del producto rellenados con la información del mismo, el nombre, descripción, categoría, precio, stock, tamaño y los 4 campos para fotos y modelos 3d. , estos estan protegidos para que no se pueda subir ningún archivo con una extensión errónea. La categoría aparecerá en una lista desplegable con su nombre, es decir, primero hay que listar las categorías, por defecto aparecerá la cual, en ese momento, pertenece al producto.
3. El administrador, le da al botón de enviar datos.
4. El sistema recoge todos los datos, y reemplaza estos mismos por los nuevos introducidos, el código del producto se volverá a generar con los mismos datos, si no se introducen las imágenes de cualquiera de las perspectivas, estas no se modificarán.

Flujo alternativo: Punto 4:

1. El sistema detecta que cualquiera de los campos no se ha introducido correctamente, este mostrará un mensaje de error, indicando de volver a rellenar los campos correctamente, el producto no se modificará.

Desactivar productos:

1. El administrador da al botón de editar un producto, en un producto en específico de todos los que estan listados.
2. El sistema le mostrará la parte del formulario para editar el producto, para editar el estado del producto.
3. El administrador puede cambiarlo (o no) y darle al botón de aceptar.
4. El sistema cambiará el estado del producto al indicado por el administrador.

Listar categorias:

1. El administrador accede al apartado de categorías.
2. El sistema le devuelve un listado de todas las categorías

Flujo alternativo: Punto 2:

1. El sistema no encuentra ninguna categoría existente en la base de datos, este mostrará un mensaje diciendo que no hay categorías y a la vez invitándonos a crear nuevas.

Crear categorías:

1. El administrador le da al botón de añadir una categoría.
2. El sistema devuelve un formulario, el cual contiene para cambiar el nombre.
3. El administrador rellena los campos y le da al botón de aceptar.
4. El sistema recoge el nombre del formulario y añade una nueva categoría con ese nombre

Flujo alternativo: Punto 3:

1. El sistema reconoce que el nombre introducido es incorrecto, en ese momento mostrará un mensaje de error y te pedirá volver a introducir los datos, no se añadirá ninguna categoría.

Editar categorías:

1. El administrador accede a la lista de todas las categorías, y le da al botón de edición de una categoría en concreto.
2. El sistema devuelve un formulario con solo un campo de nombre, este nombre es el que originalmente tiene la categoría.
3. El administrador modifica (o no) el nombre y le da al botón de aceptar.
4. El sistema recoge el nombre del formulario, y actualiza la categoría con su nuevo nombre.

Flujo alternativo: Punto 3:

1. El sistema reconoce que el nombre introducido es incorrecto, en ese momento mostrará un mensaje de error y te pedirá volver a introducir los datos, no se modificará la categoría.

Desactivar categorías:

1. El administrador accede a la lista de todas las categorías, y le da al botón de edición de una categoría en concreto.
2. El sistema devuelve el formulario de edición pero también para cambiar el estado de la categoría.
3. El administrador cambia (o no) el estado de la categoría y le da al botón de aceptar.
4. El sistema actualiza el estado de la categoría al que se ha introducido en la edición de la categoría.

Listar compradores:

1. El administrador accede al apartado de compradores.
2. El sistema le devuelve una lista de todos los compradores.

Flujo alternativo: Punto 2:

1. El sistema no encuentra ningún comprador para mostrar, así que en cambio, te muestra un mensaje de que no se ha encontrado ningún comprador.

Listar ordenes:

1. El administrador accede al apartado de órdenes.
2. El sistema muestra una lista de todas las órdenes que estén pendientes o enviadas.

Flujo alternativo: Punto 2:

1. El sistema no encuentra ningúna orden para mostrar, así que en cambio, te muestra un mensaje de que no se ha encontrado ningúna orden.

Finalizar ordenes:

1. El administrador le da al botón de editar de una de las órdenes listadas.
2. El sistema muestra un formulario para que el administrador pueda finalizar la orden y cambiar su estado a enviado.
3. El administrador cambia el estado y le da al botón de aceptar.
4. El sistema cambia el estado de la orden a enviado, y establece en el campo de fecha final, la fecha actual en la cual el administrador ha cambiado el estado a finalizado. El sistema ya no mostrará el botón para poder cambiar el estado una vez la orden este con el estado de enviado, solo las ordenes con estado pendiente seguirán teniendo este botón.

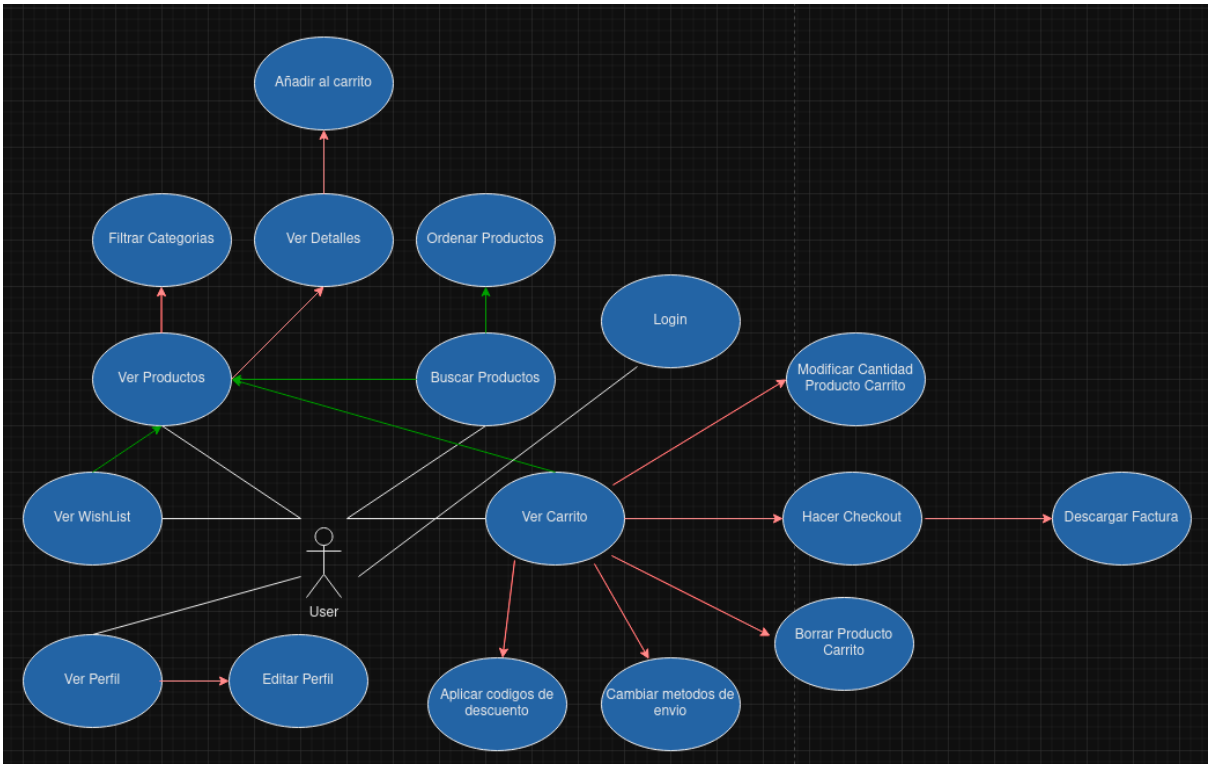
Descargar factura:

1. El administrador le da al botón de imprimir factura de una de las órdenes listadas.
2. El sistema, automáticamente, le descarga un pdf al administrador con los datos de la compra, los datos del usuario que lo ha hecho y la firma del administrador.

Flujo alternativo: Punto 2:

1. El sistema ha detectado que el administrador no dispone de firma, en vez de descargar el pdf, simplemente mostrarà un mensaje de que la factura no se podrà descargar, porque el administrador no dispone de firma para firmar la misma, invitándonos a crear una firma.

DIAGRAMA DE CASOS DE USO USER:



Login:

1. El usuario introduce los datos que quiere para loguearse.
2. El sistema relaciona los datos que ha introducido el usuario con los que hay en la base de datos sobre los usuarios registrados, si encuentra la coincidencia de la contraseña cifrada y el email, se loguea como ese usuario y lo manda a la página principal. Si hay un carrito existente en las cookies, el usuario pasará a tener ese carrito como el carrito de su usuario en la base de datos, se borrará el carrito de las cookies.

    Flujo alternativo: Punto 2:

    1. El sistema detecta que los datos introducidos no concuerdan con ningún

Ver WishList:

1. El usuario accede a su WishList mediante su botón
2. El sistema comprueba que está registrado, si lo está, muestra una lista con los productos deseados del usuario registrado.

    Flujo alternativo: Punto 2:

    1. El sistema detecta que el usuario no está registrado y nos manda al login, sugiriendo al usuario a registrarse.

Ver Perfil y editar Perfil:

1. El usuario accede a la vista de su perfil mediante su botón.
2. El sistema comprueba que está registrado, si lo está, muestra una lista con los pedidos realizados por el usuario, con la opción de eliminar los pedidos pendientes. También, muestra un formulario para cambiar algunos de sus datos.
3. El usuario rellena algunos de sus datos en el formulario y los envía esperando cambiarlos.
4. El sistema recibe los datos, si todos los datos son correctos, los cambiará.

    Flujo alternativo: Punto 2:

        1. El sistema detecta que el usuario no está registrado y nos manda al login, sugiriendo al usuario a registrarse.

    Flujo alternativo: Punto 3:

        1. El sistema detecta que los datos no estan correctamente y nada se cambiará, avisando al usuario de ello.

Ver Productos:

1. El usuario se encuentra en la página principal.
2. El sistema le devuelve una lista de los productos disponibles.

    Flujo alternativo: Punto 2:

        1. El sistema detecta que no hay productos para mostrar, mostrando un mensaje que nos dice que no hay productos disponibles.

Buscar productos:

1. El usuario accede a la ventana de buscar productos mediante un botón.
2. El sistema le muestra una ventana con todo el listado de productos y una barra de búsqueda.
3. El usuario escribe algo en el buscador.
4. El sistema le muestra al usuario los productos que tengan las letras coincidentes en el nombre.

    Flujo alternativo: Punto 2:

        1. El sistema detecta que no hay productos para mostrar, mostrando un mensaje que nos dice que no hay productos disponibles.

Ordenar productos:

1. El usuario accede a la ventana de buscar productos mediante un botón.
2. El sistema le muestra una ventana con todo el listado de productos y una barra de búsqueda.
3. El usuario, mediante un seleccionable, dice cómo quiere ordenar los productos que aparecen según su precio.
4. El sistema cambia el formato en el que se muestran los datos para mostrarlos en el orden que especifica el usuario.

Filtrar categorias:

1. El usuario se encuentra en la página principal.
2. El sistema muestra un listado de las categorías disponibles.
3. El usuario hace clic en una de las categorías.
4. El sistema, cambia el formato donde se listan los productos, para ahora solo mostrar los pertenecientes a la categoría seleccionada por el usuario

    Flujo alternativo: Punto 2:

        1. El sistema detecta que no hay categorías para mostrar, así que no aparecerá el listado de categorías.

Ver detalles:

1. El usuario accede al detalle del producto dando clic sobre un producto que se muestra en la página principal.
2. El sistema redirige a la página del detalle del producto específico que ha seleccionado el usuario, donde este podrá ver los datos de este.

Añadir al carrito:

1. El usuario, en el detalle del producto, selecciona la talla que quiere y le da al botón de añadir al carrito.
2. El sistema, dependiendo si el usuario está logueado o no, guarda el producto en la base de datos o en las cookies de la página.

    Flujo alternativo: Punto 2:

        1. El sistema detecta que no hay stock de ese producto para añadirlo al carrito, te muestra un mensaje de error y vuelve al detalle del producto.

Ver carrito:

1. El usuario, accede mediante el botón de acceso del carrito al carrito.
2. El sistema cambia de ventana a la del carrito, si está logueado, mostrará los datos del carrito de ese usuario de la base de datos, si no está logueado, mostrará los datos del carrito que haya en las cookies de este navegador.

Modificar cantidad de carrito:

1. El usuario le da al botón de sumar o restar del carrito de uno de los productos.
2. El sistema primero ve si está o no registrado el usuario, si no lo está, accedera a las cookies y mirara si se puede restar (no es menos de 1) o si se puede sumar (no se pasa la cantidad del stock disponible del producto), y entonces sumara o restara de los datos de las cookies, si esta logueado, hace el mismo procedimiento pero haciendo los cambios directamente al carrito del usuario correspondiente de la base de datos.

Borrar producto del carrito:

1. El usuario le da al botón de borrar del carrito de uno de los productos.
2. El sistema detecta si el usuario está logueado, si lo está, borrara ese producto de los datos del carrito del usuario correspondiente de la base de datos, si no lo está, borrara ese producto de la lista del carrito de las cookies.

Cambiar metodos de envio:

1. El usuario accede a la página del carrito mediante su botón en la página principal, o cualquier otra.
2. El sistema, al cargar la página del carrito, le mostrará un seleccionable según los métodos de envío que existan en la base de datos.
3. El usuario elige uno de ellos.
4. El sistema, le suma al precio total de los productos del carrito el valor de precio que tenga ese método de envío especificado en la base de datos.

Aplicar códigos de descuento:

1. El usuario, en la página del carrito con varios productos, accede al input para poder añadir un código de descuento.
2. El sistema, mira si el código introducido por el usuario coincide con alguno de los códigos de la base de datos, si lo encuentra, revisa que el usuario no haya introducido antes otro código de descuento para no acumularlos y le aplicará el valor de ese código de descuento al precio total.

Hacer checkout:

1. El usuario le da a comprar el carrito.

2. El sistema comprueba que el usuario esté validado, también que el carrito no esté sin productos, si encuentra productos, mira la cantidad de cada uno de ellos y comprueba que haya stock, si hay stock de todos los productos solicitados por el usuario, la compra será un éxito y se harán los cambios en la base de datos correspondientes para que el carrito pase a convertirse en una orden de compra pendiente.

   Flujo alternativo: Punto 2:

   1. El sistema detecta que el usuario no está validado, simplemente se le envia al login.

   Flujo alternativo: Punto 2:

   1. El sistema detecta que el carrito no tiene productos, así que el botón de compra no hará nada.

   Flujo alternativo: Punto 2:

   1. El sistema detecta que los productos del carrito superan el stock disponible, así que no se hará la compra y se avisará al usuario que reduzca el stock de su compra o espere a que se recargue ese producto.

Descargar factura:

1. El usuario, al comprar el carrito, le sale la opción de descargar la factura de la compra.
2. El sistema le devuelve al usuario un pdf con los datos de la compra.

Carrito:

-En la tabla shopping, un usuario, puede comprar un producto, si este usuario no dispone de un carrito en estado de ‘shipped’, se creará un nuevo carrito con el status mencionado y el useremail correspondiente al usuario, todos los demás valores estaran vacíos de momento.

-Al momento, también se creará una instancia de inCart en la que se añadira, en el valor shop referenciando al identificador del carrito, el producto añadido (su ID), la size escogida y la cantidad siempre se añadirá en 1.

-Después, hay que tener en cuenta que si se añade al carrito otro producto, simplemente se hará otra instancia inCart señalando al mismo y los datos seleccionados en el detalle del producto, ahora, si se añade el mismo producto, pero con otro tamaño, se creará una instancia nueva como si fuese un producto nuevo para el carrito, pero si es el mismo producto y también el tamaño es el mismo, simplemente ya habra una instancia inCart existente a la cual se le añadira +1 en el amount.

-Ahora, en el mismo carrito, si se sube o se baja el amount dinámicamente, no puede bajar de 1 y no puede subir más de el stock disponible, al momento se irán también en la base de datos cambiando el amount en inCart.

-Si se elimina el producto, simplemente se elimina la instancia inCart, pero si se eliminan todos los productos, el carrito se borrara de la base de datos entero.

-Si el usuario no esta validado, todo el proceso explicado será similar, pero en vez de hacer las consultas de la base de datos, los datos se irán pasando a javascript y se guardaran en las cookies y podrá hacer exactamente lo mismo que como si fuese la base de datos, una vez el usuario quiere comprar el carrito, se le muestra el login, inicia sesión y si con el usuario con el que inicia no tiene ningún carrito en shipped, se harán las consultas necesarias para crear en la base de datos el carrito que ha estado creado en las cookies, sino, simplemente iniciara y el carrito de las cookies se eliminará.