EasyStore

Código Fuente

Resumen

EasyStore es una aplicación que permite gestionar inventarios y finanzas para pequeños negocios y/o emprendimientos. Permite manejar proveedores, stock de productos, reposición, etc. Adicionalmente también permite manipular las finanzas, saber los ingresos, los gastos, calcular el balance.

Tecnologías/Herramientas utilizadas

Para realizar este proyecto he utilizado Java como lenguaje principal, en conjunto con la librería JDBC para SQLite (para poder manejar los datos de la aplicación).

Para realizar los diagramas UML he utilizado PlanTextUML un software gratuito que mediante un lenguaje de marcado permite generar los diagramas. Para el diagrama de base de datos he utilizado dbdiagram.io que permite, de nuevo, por el mismo sistema generar diagramas de base de datos con sus relaciones, constrains, etc.

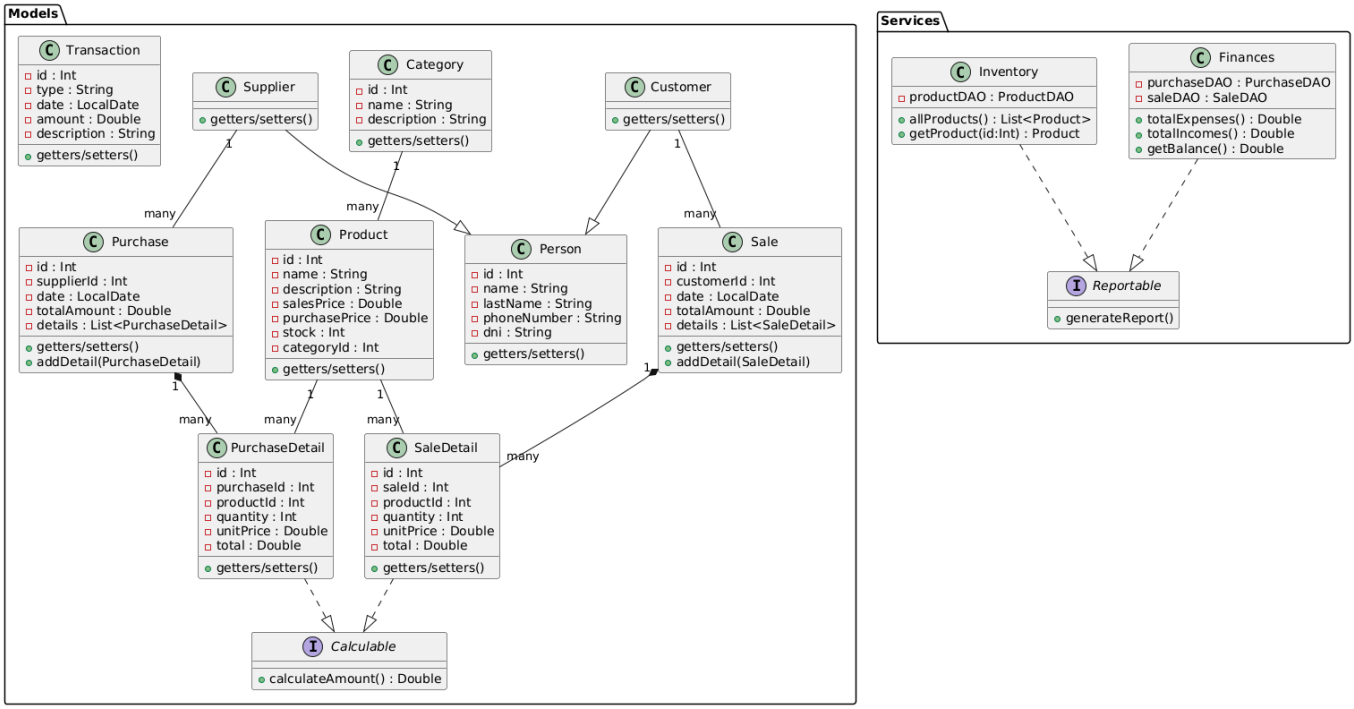
Para el desarrollo utilice Apache Netbeans y Nvim. Adicionalmente algunas otras herramientas que utilicé fueron git y github para subir el código al repositorio. Así como TablePlus para inspeccionar bases de datos. Para hacer el informe utilice Obsidian con markdown.

Adicionalmente como recursos durante el desarrollo he utilizado Gemini como asistente de IA, YouTube y diversos blogs sobre arquitectura o diseño de sistemas.

Funcionamiento de la aplicación

La aplicación se organiza mediante el diseño MVC (Model View Controller) en el cual se logra separar cada capa de la vista al usuario, protegiendo así a los modelos, la base de datos y los controladores que se encargan de ello juustamente.

Diagrama UML



Organización de paquetes

Los paquetes se organizan de la siguiente manera:

- **easystore.iu:** almacena el archivo Main que es el punto de entrada de la aplicación.
- **easystore.dao:** almacena los archivos que representa cada DAO (Data Access Object)
- **easystore.db:** almacena el archivo de SQLite donde está la base de datos como tal
- **easystore.model:** almacena los archivos que representan los modelos de la base de datos
- **easystore.service:** almacena los archivos que tienen la lógica de negocio
- **easystore.exceptions:** almacena las excepciones personalizadas de la aplicación.

¿Qué son los DAO?

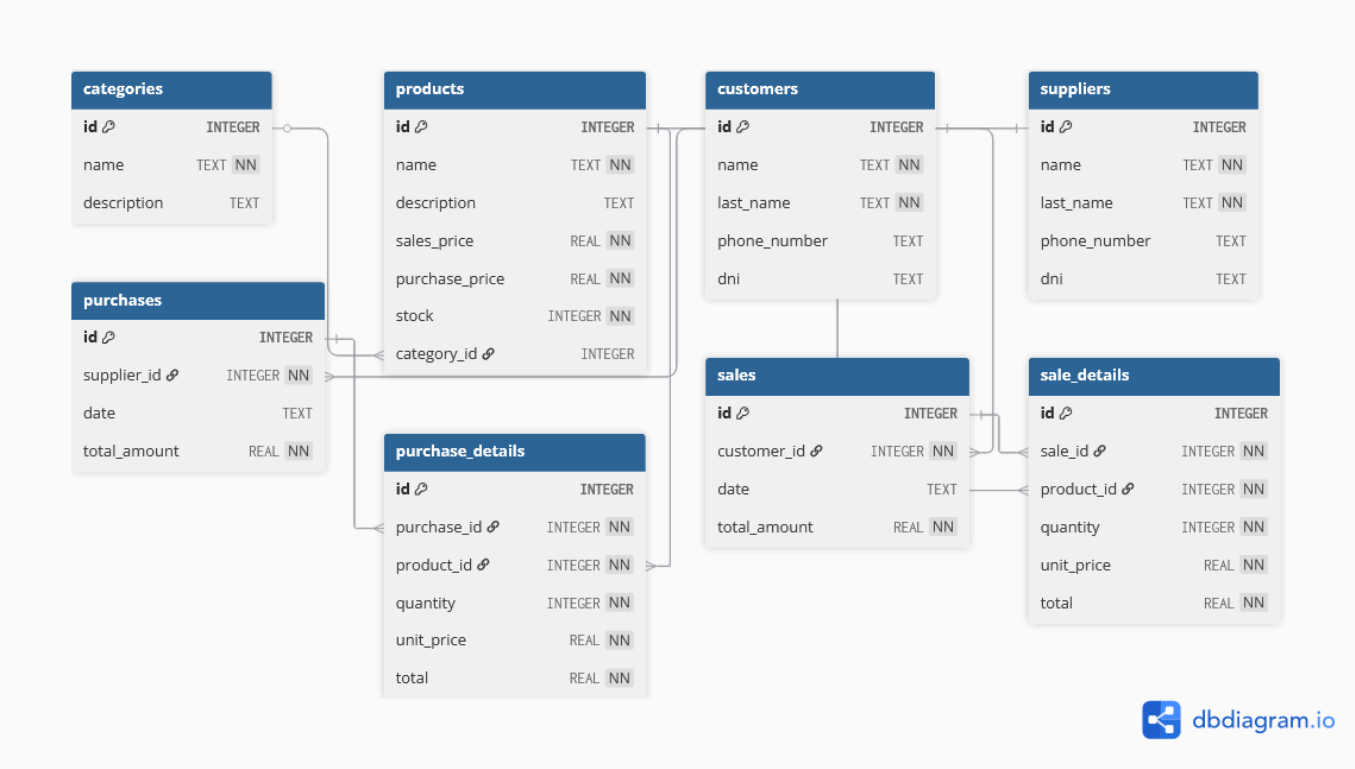
"El patrón Arquitectónico Data Access Object (DAO), el cual permite separar la lógica de acceso a datos de los Objetos de negocios (Bussines Objects), de tal forma que el DAO encapsula toda la lógica de acceso de datos al resto de la aplicación." Fragmento extraído de: Patrones de Arquitectura

¿Qué es JavaDocs?

Los comentarios Javadoc son comentarios especiales que se utilizan para proporcionar una descripción del elemento de código situado debajo de ellos. Comienzan con /* *y terminan con* / y pueden contener etiquetas marcadas con @ con metadatos específicos. Esto ayuda a la legibilidad y entendimiento del código para el propio programador,

pero sobre todo para los demás programadores. Además JavaDocs permite generar una referencia del programa en HTML.
Fragmento extraído de: JetBrains

Diagrama de base de datos



El diagrama de base de datos representa las diferentes tablas y las relaciones entre estas. A continuación el código SQL.

```sql
-- Sentencia para activar las foreing keys que por defecto se encuentran desacticadas en SQLite
PRAGMA foreign_keys = ON;

-- Tabla de categorías: almacena las categorías
CREATE TABLE categories (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  name TEXT UNIQUE NOT NULL,
  description TEXT
);

-- Tabla de productos: almacena los productos, tiene una referencia a la categoría a la que pertenece
CREATE TABLE products (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  name TEXT NOT NULL,
  description TEXT,
```

```
  sales_price REAL NOT NULL,
  purchase_price REAL NOT NULL,
  stock INTEGER NOT NULL DEFAULT 0,
  category_id INTEGER,
  FOREIGN KEY (category_id) REFERENCES categories (id) ON DELETE SET NULL --
La sentencia ON DELETE SET NULL permite que si se elimina una categoría se
setee la categoría del producto en NULL y no se elimine el mismo, esto con
el fin de no perder productos cargados
);

-- Tabla de clientes: almacena los clientes
CREATE TABLE customers (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  name TEXT NOT NULL,
  last_name TEXT NOT NULL,
  phone_number TEXT,
  dni TEXT UNIQUE
);

-- Tabla de proveedores: almacena los proveedores
CREATE TABLE suppliers (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  name TEXT NOT NULL,
  last_name TEXT NOT NULL,
  phone_number TEXT,
  dni TEXT UNIQUE
);

-- Tabla de compras: almacena las compras realizadas, tiene referencia a la
tabla de proveedores
CREATE TABLE purchases (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  supplier_id INTEGER NOT NULL,
  date TEXT,
  total_amount REAL NOT NULL,
  FOREIGN KEY (supplier_id) REFERENCES suppliers (id)
);

-- Tabla de detalles de compra: almacena los detalles de cada compra, tiene
referencia a la tabla de compras y productos
CREATE TABLE purchase_details (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  purchase_id INTEGER NOT NULL,
  product_id INTEGER NOT NULL,
  quantity INTEGER NOT NULL,
  unit_price REAL NOT NULL,
```

```
  total REAL NOT NULL,
  FOREIGN KEY (purchase_id) REFERENCES purchases (id) ON DELETE CASCADE, --
La sentencia ON DELETE CASCADE permite que si se elimina una compra sus
detalles también se van a borrar, ya que no tendría sentido almacenar los
detalles de una compra que ya no existe
  FOREIGN KEY (product_id) REFERENCES products (id)
);

-- Tabla de ventas: almacena las ventas realizadas, tiene referencia a la
tabla de clientes
CREATE TABLE sales (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  customer_id INTEGER NOT NULL,
  date TEXT,
  total_amount REAL NOT NULL,
  FOREIGN KEY (customer_id) REFERENCES customers (id)
);

-- Tabla de detalle de ventas: almacena las ventas realizadas, tiene
referencia a la tabla de ventas y productos
CREATE TABLE sale_details (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  sale_id INTEGER NOT NULL,
  product_id INTEGER NOT NULL,
  quantity INTEGER NOT NULL,
  unit_price REAL NOT NULL,
  total REAL NOT NULL,
  FOREIGN KEY (sale_id) REFERENCES sales (id) ON DELETE CASCADE,
  FOREIGN KEY (product_id) REFERENCES products (id)
);
```

Como se puede observar, la estructura de tablas y sus relaciones, es bastante sencilla. La lógica de la aplicación permite crear una sola conexión a la base de datos para no crear muchas instancias y que esto pueda ocasionar problemas, este patrón es conocido también como Singleton.

"El patrón de diseño Singleton recibe su nombre debido a que sólo se puede tener una única instancia para toda la aplicación de una determinada clase, esto se logra restringiendo la libre creación de instancias de esta clase mediante el operador new e imponiendo un constructor privado y un método estático para poder obtener la instancia."
Fragmento extraído de: Patrones de diseño