Imagen que contiene imágenes prediseñadas

Descripción generada con confianza muy alta 

ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

2S-2018

*DISEÑO DE SOFTWARE*

*Proyecto: “PoliVentas”*

**INTEGRANTES:**

* Aguirre Larrosa Stefanny Briggitte
* Bermúdez Moreira Karen Monserrat
* Figueroa Villacreses Galo Xavier
* Pincay Jiménez Rosa María

PARALELO 1

PROFESOR: DAVID JURADO

GUAYAQUIL – ECUADOR

ÍNDICE

[DESCRIPCIÓN DEL SISTEMA 3](#_Toc535164079)

[JUSTIFICACIÓN DE PATRONES DE DISEÑO UTILIZADOS 4](#_Toc535164080)

# DESCRIPCIÓN DEL SISTEMA

PoliVentas es una plataforma diseñada para ser utilizada por estudiantes como una aplicación de compra y venta de artículos varios, la cual, de manera general muestra cuales son los productos más vendidos donde el usuario, ya sea visitante o un estudiante registrado en el sistema tiene la posibilidad de realizar la compra de alguno de ellos y donde en caso de ser un usuario público, este pueda registrarse a través de un formulario, también existe un usuario administrador que es el responsable de agregar usuarios, y productos del sistema. Este sistema, el cual funciona mediante una conexión a una base de datos relacional, permite además realizar una compra a través de 2 métodos de pago actualmente, y debe a su vez permitir el ingreso de nuevos roles y métodos de pago a futuro.

# JUSTIFICACIÓN DE PATRONES DE DISEÑO UTILIZADOS

En este proyecto se emplearon los siguientes patrones de diseño:

* **Singleton**

Este patrón se caracteriza porque una determinada clase se instancia una sola vez durante la ejecución del programa, en el entorno de la aplicación, la conexión con la base de datos debería tener ese comportamiento, al iniciar el programa, este se conecta a la base de datos, ya que todas la opciones del sistema dependen de esta, debería haber alguna forma global que permita acceder a dicha conexión, además ninguna clase debería poder crear conexiones nuevas ya que esto sería ineficiente, por este motivo, implementar este patrón de diseño sería muy útil en esta situación.

* **Strategy**

Este patrón se aplica en funciones o servicios determinados que se pueden realizar de distintas maneras, el pago de los productos es receptado en efectivo o por medio de la plataforma, por tal motivo se ha usado Strategy para dichos métodos de pago, ya que este patrón permite al cliente elegir el según sus preferencias. Es de gran utilidad, puesto que el sistema debe permitir el ingreso de nuevos métodos de pago a futuro, lo cual solo cambia el algoritmo de su implementación sin perjudicar otras funcionalidades existentes.

* **Observer**

Este patrón que tiene la característica de informar cuando un objeto ha cambiado su estado, es útil para modelar la situación en el que el cliente paga alguno de sus pedidos y el vendedor es notificado para poder establecer el lugar de entrega de los productos solicitados, en base a esto se puede distinguir que:

* La clase a observar es la clase **Pedido,** cuando su estado cambie a *pagado,* notificará a sus observadores lo ocurrido.
* La clase que observa los cambios es la clase **Vendedor**

De esta forma se puede relacionar de manera eficiente la clase pedido con vendedor y se logra un bajo acoplamiento.