

# **UNIVERSIDAD DE LAS FUERZAS ARMADAS**

**“ESPE”**



## **PROGRAMACIÓN ORIENTADA A OBJETOS**

### **Actividad Experimental N° 02**

Sistema de Gestión de Inventario con MVC.

#### **Tema de la Actividad:**

Implementación de un sistema que gestione el inventario de una tienda utilizando el patrón

Modelo Vista Controlador (Caso Práctico)

**Integrantes Grupo # 06**

**NRC: 1322**

Mónica Carolina Angamarca Cela.

Vanessa Fernanda Chiriguaya Ruales.

Luis Alejandro Sánchez Durán.

**Quito, 03 de febrero del 2025**

## **1.- Introducción**

En el ámbito económico actual, la matriz productiva exige cada vez más innovaciones tecnológicas que sean de utilidad para mejorar el desarrollo de economía, en donde la industria local sea reconocida a nivel internacional. Es por ello que las TIC's se ven reflejadas en las empresas, ya que, si bien existen múltiples formas de solucionar el típico problema del manejo manual de la información, esto se convierte en una deficiencia por ser sistemas estáticos.

Por lo cual, el presente informe dará a conocer una solución práctica y sencilla para gestionar un inventario implementando el conocimiento adquirido en la programación orientada a objetos, utilizando para su desarrollo el patrón de diseño MVC.

El patrón de diseño MVC, es ampliamente utilizado en el desarrollo de software continuo que divide la lógica de modelado de acceso del motor de base de datos en múltiples capas, desde la capa de presentación hasta la lógica de negocio, llamada controlador. El modelo interactúa ampliamente con el motor de base de datos, que incluyen los métodos de acceso a entidades (clases que hacen referencia a tablas en el motor de base de datos). La vista es la implementación de la interacción visual con el usuario final, en la cual se realizará la entrada de datos que van a suministrar la información dentro de la base de datos.

Estas características en el sistema de gestión de una tienda, permite mejorar la organización del código, facilita la reutilización de componentes y simplificar futuras modificaciones. Como resultado, se convierte en una pieza útil en entornos comerciales donde los cambios son constantes en productos, interfaz o stock.

Tomando en cuenta que actualmente existen varios proyectos exitosos unos más utilizados que otros, ya sea por las funcionalidades extras, la usabilidad o vista del programa es más agradable, recomendaciones de otras personas, entre otras. Es por ello que en este informe se detallarán los pasos a seguir desde el desarrollo del UML, que es el primer paso, hasta la ejecución del código.

## **2.- Objetivos:**

### **2.1.- Objetivo general**

Crear y codificar un sistema que gestione el inventario de una tienda utilizando el patrón Modelo Vista Controlador (MVC). Permitiendo funcionalidades como registrar y consultar productos, actualizar y calcular el valor total del inventario, entre otros implementos que serán visualizados mediante la ejecución del programa.

### **2.2.- Objetivos específicos**

- Diseñar el UML en base a las funcionalidades que se desea en el sistema de gestión.
- Diseñar un código que permita gestionar el inventario de una tienda utilizando el patrón MVC.
- Explicar la funcionalidad del programa y los beneficios de su utilización.

## **3.- Marco Teórico**

En una empresa, los sistemas de gestión de inventarios son una de las herramientas más importantes a nivel de logística, debido a su utilidad en todo lo referente a la eficiencia de procesos, por lo que cumplen un papel destacado a la hora de garantizar la administración de cualquier negocio.

Esta herramienta permite conocer y gestionar los aspectos relacionados al inventario desde el nivel de producción hasta el comercio minorista, el almacén y los procesos logísticos.

Según el blog DispatchTrack (2022), los tipos de sistemas de gestión de inventarios suelen estar clasificados en cuatro tipos, los cuales son:

- *Sistemas de inventario perpetuo:* Son comunes para el caso de manejo de cantidades pequeñas o medianas de stock. Cuenta con una base de datos fija que se va actualizando según los procesos logísticos
- *Sistemas de inventarios periódicos:* No operan con información en tiempo real, sino que realizan actualizaciones en intervalos definidos, reemplazando los datos

anteriores por los más recientes. Funcionan con un inventario inicial y final, donde el final de un período se convierte en el inicial del siguiente.

- *Sistemas de código de barras*: Utilizan códigos de barras para identificar y rastrear productos, ofreciendo mayor precisión y eficiencia en comparación con sistemas manuales.
- *Sistemas de identificación por radiofrecuencia (RFID)*: Basados en tecnología RFID, permiten el seguimiento automático del inventario en grandes volúmenes, siendo altamente eficaces en entornos logísticos.

Además de estos métodos, existen softwares especializados que optimizan la gestión del inventario, como:

- *Factusol*: Programa especializado en control de stock en toda la cadena de suministro.
- *Odoo Inventory*: ERP flexible con módulo de inventario opcional.
- *ABC Inventory*: Solución gratuita para la gestión de inventarios.
- *Holded*: ERP en la nube, intuitivo y fácil de usar.

El patrón de diseño MVC, es ampliamente utilizado en el desarrollo de software continuo que divide la lógica de modelado de acceso del motor de base de datos en múltiples capas, desde la capa de presentación hasta la lógica de negocio, llamada controlador. El modelo interactúa ampliamente con el motor de base de datos, que incluyen los métodos de acceso a entidades (clases que hacen referencia a tablas en el motor de base de datos). La vista es la implementación de la interacción visual con el usuario final, en la cual se realizará la entrada de datos que van a suministrar la información dentro de la base de datos (Salomón, 2021).

Estas características en el sistema de gestión de una tienda, permite mejorar la organización del código, facilita la reutilización de componentes y simplificar futuras modificaciones. Como resultado, se convierte en una pieza útil en entornos comerciales donde los cambios son constantes en productos, interfaz o stock.

Entonces, teniendo claro los conceptos anteriormente descritos, a continuación se realizará el proceso para que este programa sea un sistema que gestione el inventario de una tienda utilizando el patrón Modelo Vista Controlador (MVC). El cual realizará las siguientes

funciones:

**Modelo (Producto y Inventario):**

- Registrar Productos: Implementado en la clase Inventario como un método para agregar instancias de Producto a una lista.
- Actualizar Inventario: Método en la clase Inventario que modifica la cantidad de un producto.
- Consultar Productos: Método en la clase Inventario que devuelve la lista de productos.
- Calcular el Valor Total del Inventario: Método en la clase Inventario que itera sobre los productos y calcula el total.

**Vista (VistaConsola o GUI):**

- Mostrar la lista de productos al usuario.
- Presentar el valor total del inventario calculado por el modelo.
- Solicitar datos al usuario para registrar productos o actualizar inventario.

**Controlador (ControladorInventario):**

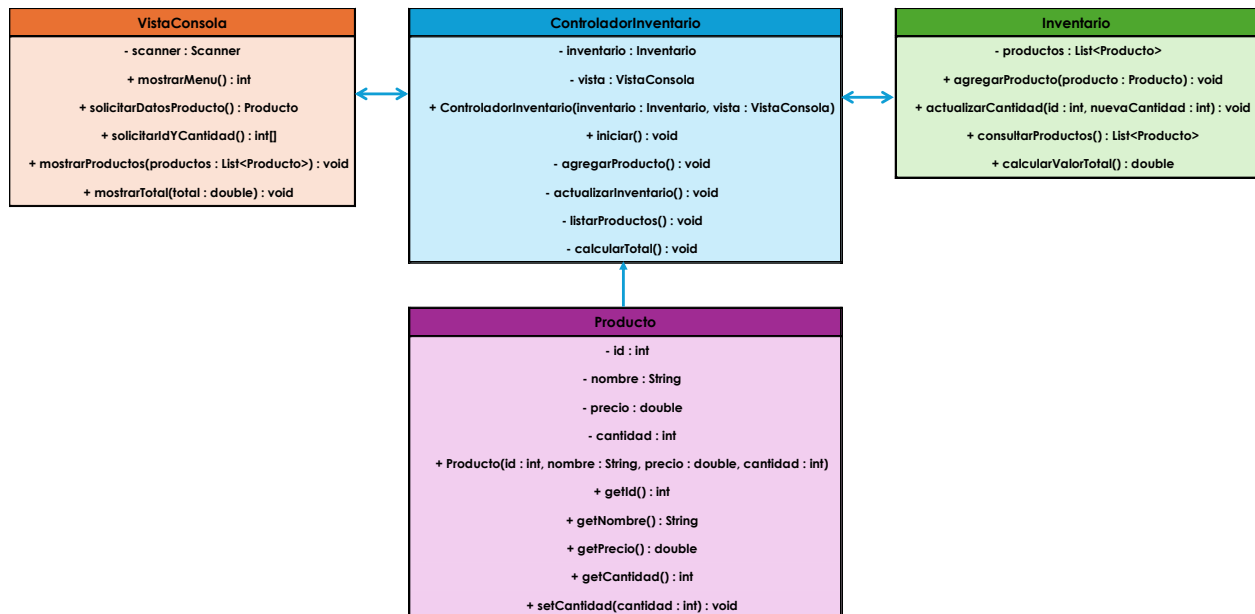
- Coordina entre la Vista y el Modelo.
- Recibe entradas del usuario desde la Vista, las procesa y las envía al Modelo.
- Devuelve la información del Modelo a la Vista para mostrarla al usuario.

**4.- Metodología – Exposición:**

El grupo N° 06 realizó la implementación de un sistema que gestiona el inventario de una tienda de ropa “Su punto ideal de compra” utilizando el patrón Modelo Vista Controlador.

En este código podemos observar la implementación de un sistema de inventario simple que sigue el patrón MVC, facilitando de esta manera la separación de responsabilidades entre lógica de datos, interfaz de usuario y control de flujo de ejecución.

Empezamos con la elaboración del Diagrama de clases UML que muestre las entidades principales (productos, inventario, precio, cantidad, etc.) y sus relaciones.



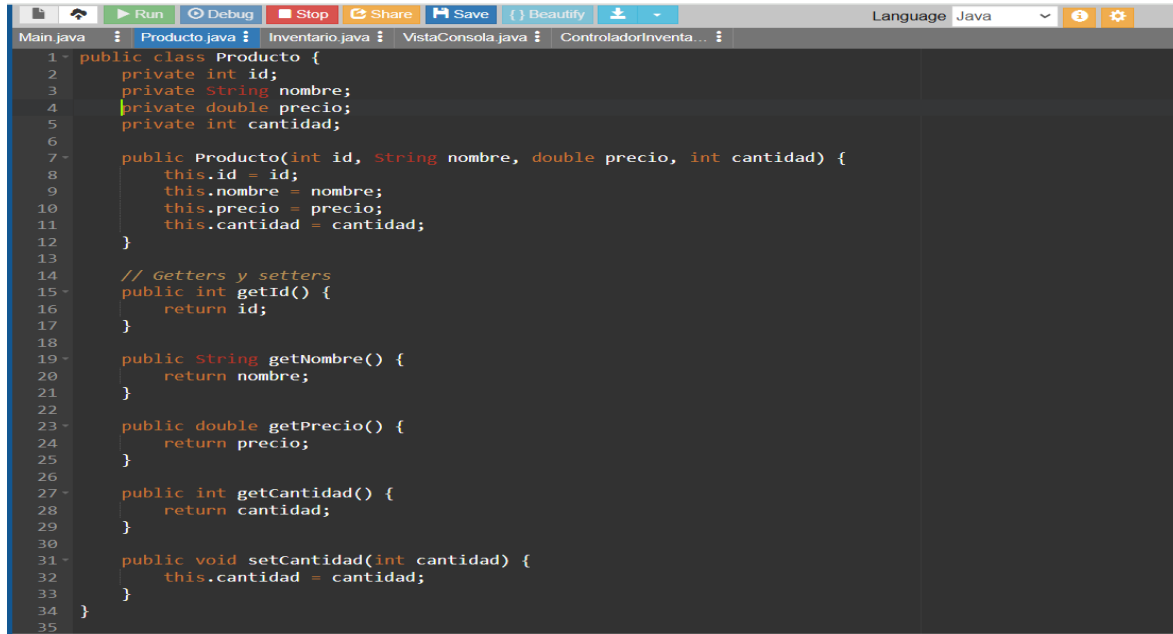
Una vez elaborado el UML se procedera a la creación del código, en el que nos apoyaremos del Lenguaje de programación Java, en el cual podremos mostrar la descripción de la su estructura del código, separando claramente Modelo, Vista y Controlador.

## Clase Main

```

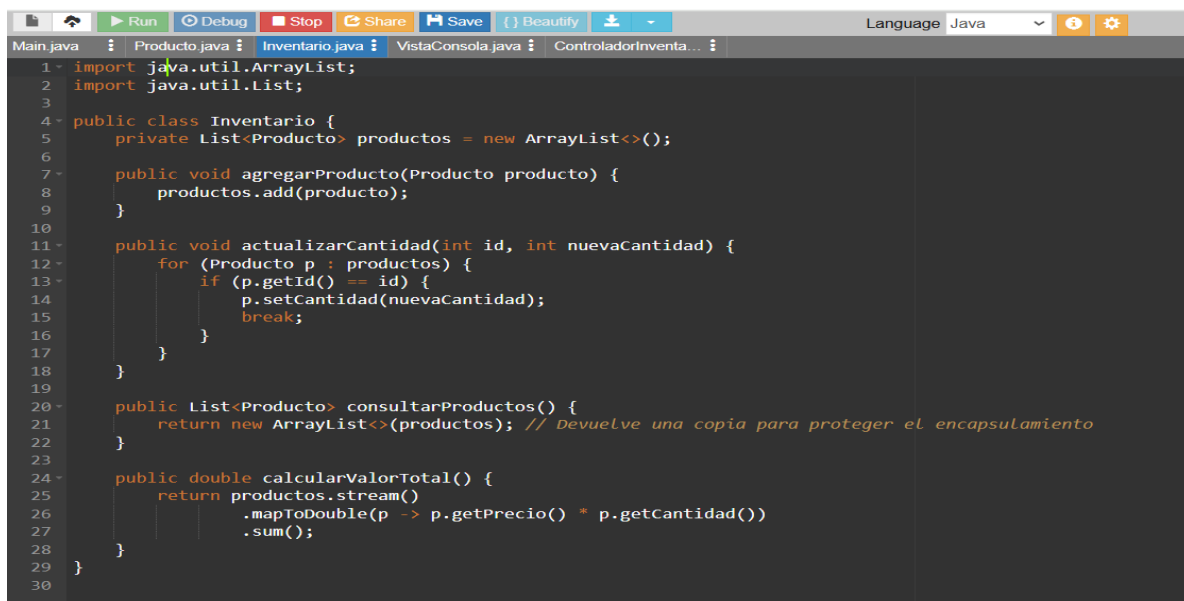
1 public class Main {
2     public static void main(String[] args) {
3         Inventario inventario = new Inventario();
4         VistaConsola vista = new VistaConsola();
5         ControladorInventario controlador = new ControladorInventario(inventario, vista);
6         controlador.iniciar();
7     }
8 }
9
  
```

## Clase Producto



```
1- public class Producto {
2-     private int id;
3-     private String nombre;
4-     private double precio;
5-     private int cantidad;
6-
7-     public Producto(int id, String nombre, double precio, int cantidad) {
8-         this.id = id;
9-         this.nombre = nombre;
10-        this.precio = precio;
11-        this.cantidad = cantidad;
12-    }
13-
14-    // Getters y setters
15-    public int getId() {
16-        return id;
17-    }
18-
19-    public String getNombre() {
20-        return nombre;
21-    }
22-
23-    public double getPrecio() {
24-        return precio;
25-    }
26-
27-    public int getCantidad() {
28-        return cantidad;
29-    }
30-
31-    public void setCantidad(int cantidad) {
32-        this.cantidad = cantidad;
33-    }
34- }
35
```

## Clase Inventario



```
1- import java.util.ArrayList;
2- import java.util.List;
3-
4- public class Inventario {
5-     private List<Producto> productos = new ArrayList<>();
6-
7-     public void agregarProducto(Producto producto) {
8-         productos.add(producto);
9-     }
10-
11-     public void actualizarCantidad(int id, int nuevaCantidad) {
12-         for (Producto p : productos) {
13-             if (p.getId() == id) {
14-                 p.setCantidad(nuevaCantidad);
15-                 break;
16-             }
17-         }
18-     }
19-
20-     public List<Producto> consultarProductos() {
21-         return new ArrayList<>(productos); // Devuelve una copia para proteger el encapsulamiento
22-     }
23-
24-     public double calcularValorTotal() {
25-         return productos.stream()
26-             .mapToDouble(p -> p.getPrecio() * p.getCantidad())
27-             .sum();
28-     }
29- }
30
```

## Clase VistaConsola

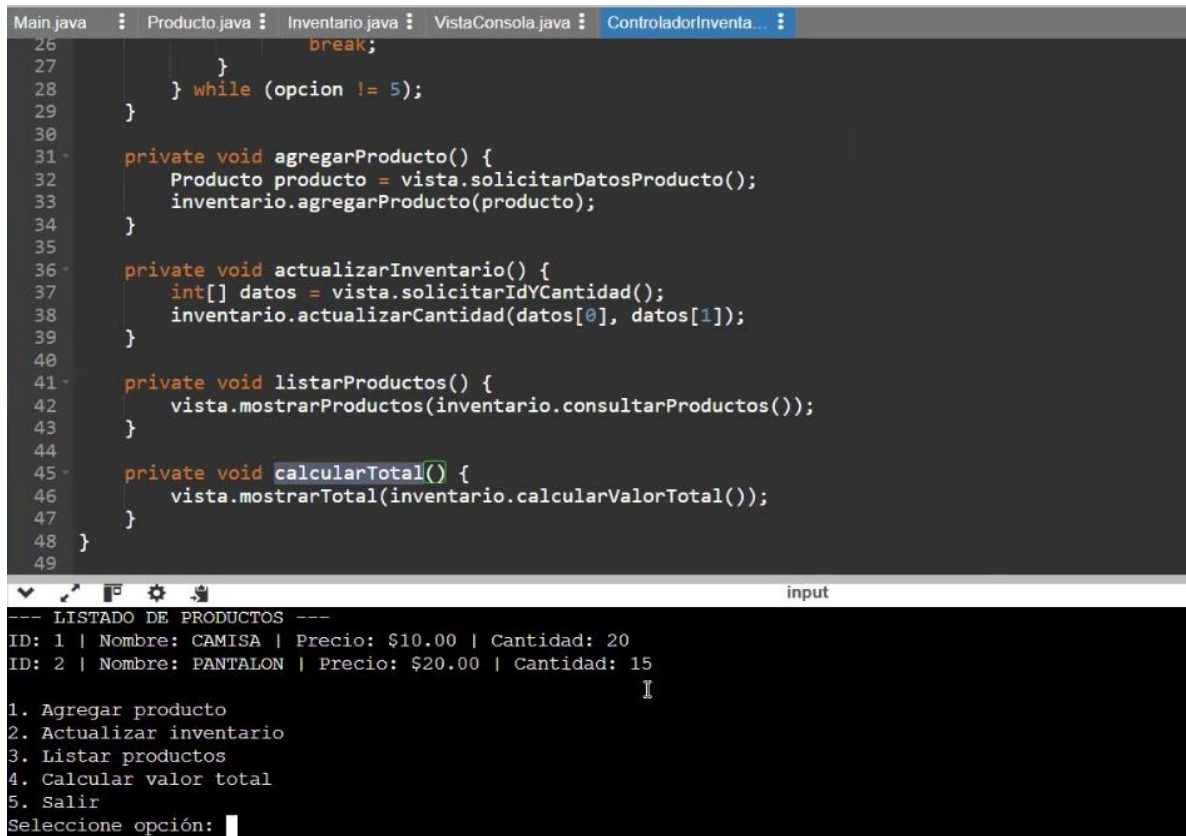
```
1- import java.util.List;
2- import java.util.Scanner;
3-
4- public class VistaConsola {
5-     private Scanner scanner = new Scanner(System.in);
6-
7-     public int mostrarMenu() {
8-         System.out.println("\n1. Agregar producto");
9-         System.out.println("2. Actualizar inventario");
10-        System.out.println("3. Listar productos");
11-        System.out.println("4. Calcular valor total");
12-        System.out.println("5. Salir");
13-        System.out.print("Seleccione opción: ");
14-        return scanner.nextInt();
15-    }
16-
17-    public Producto solicitarDatosProducto() {
18-        System.out.print("ID: ");
19-        int id = scanner.nextInt();
20-        System.out.print("Nombre: ");
21-        String nombre = scanner.next();
22-        System.out.print("Precio: ");
23-        double precio = scanner.nextDouble();
24-        System.out.print("Cantidad: ");
25-        int cantidad = scanner.nextInt();
26-        return new Producto(id, nombre, precio, cantidad);
27-    }
28-
29-    public int[] solicitarIdYCantidad() {
30-        System.out.print("ID del producto: ");
31-        int id = scanner.nextInt();
32-        System.out.print("Nueva cantidad: ");
33-        int cantidad = scanner.nextInt();
34-        return new int[]{id, cantidad};
35-    }
36-
37-    public void mostrarProductos(List<Producto> productos) {
38-        System.out.println("\n--- LISTADO DE PRODUCTOS ---");
39-        for (Producto p : productos) {
40-            System.out.printf("ID: %d | Nombre: %s | Precio: %.2f | Cantidad: %d\n",
41-                               p.getId(), p.getNombre(), p.getPrecio(), p.getCantidad());
42-        }
43-    }
44-
45-    public void mostrarTotal(double total) {
46-        System.out.printf("\nValor total del inventario: %.2f\n", total);
47-    }
48-}
```

## Clase ControladorInventario

```
1- public class ControladorInventario {
2-     private Inventario inventario;
3-     private VistaConsola vista;
4-
5-     public ControladorInventario(Inventario inventario, VistaConsola vista) {
6-         this.inventario = inventario;
7-         this.vista = vista;
8-     }
9-
10-    public void iniciar() {
11-        int opcion;
12-        do {
13-            opcion = vista.mostrarMenu();
14-            switch (opcion) {
15-                case 1:
16-                    agregarProducto();
17-                    break;
18-                case 2:
19-                    actualizarInventario();
20-                    break;
21-                case 3:
22-                    listarProductos();
23-                    break;
24-                case 4:
25-                    calcularTotal();
26-                    break;
27-            }
28-        } while (opcion != 5);
29-    }
30-
31-    private void agregarProducto() {
32-        Producto producto = vista.solicitarDatosProducto();
33-        inventario.agregarProducto(producto);
34-    }
35-
36-    private void actualizarInventario() {
37-        int[] datos = vista.solicitarIdYCantidad();
38-        inventario.actualizarCantidad(datos[0], datos[1]);
39-    }
40-
41-    private void listarProductos() {
42-        vista.mostrarProductos(inventario.consultarProductos());
43-    }
44-
45-    private void calcularTotal() {
46-        vista.mostrarTotal(inventario.calcularValorTotal());
47-    }
48-}
```



## Ejecución del programa



The screenshot shows an IDE with a Java project. The top pane displays the code for `ControladorInventa...` (ControladorInventario.java). The code includes a `while` loop for a menu, and several private methods: `agregarProducto()`, `actualizarInventario()`, `listarProductos()`, and `calcularTotal()`. The bottom pane shows the console output, which displays a list of products and a menu for the user to select an action.

```
26         break;
27     }
28     } while (opcion != 5);
29 }
30
31 private void agregarProducto() {
32     Producto producto = vista.solicitarDatosProducto();
33     inventario.agregarProducto(producto);
34 }
35
36 private void actualizarInventario() {
37     int[] datos = vista.solicitarIdYCantidad();
38     inventario.actualizarCantidad(datos[0], datos[1]);
39 }
40
41 private void listarProductos() {
42     vista.mostrarProductos(inventario.consultarProductos());
43 }
44
45 private void calcularTotal() {
46     vista.mostrarTotal(inventario.calcularValorTotal());
47 }
48 }
49
```

```
--- LISTADO DE PRODUCTOS ---
ID: 1 | Nombre: CAMISA | Precio: $10.00 | Cantidad: 20
ID: 2 | Nombre: PANTALON | Precio: $20.00 | Cantidad: 15
1. Agregar producto
2. Actualizar inventario
3. Listar productos
4. Calcular valor total
5. Salir
Seleccione opción: 
```

## 5.- Recomendaciones:

Una vez que hemos culminado con nuestra implementación podemos recomendar que se debe:

Comprender los fundamentos: Esta es la base de todo. Si no entiendes por qué existe MVC y cómo funciona la separación de responsabilidades, será muy difícil que puedas aplicarlo correctamente. Dedica tiempo a estudiar los conceptos clave y asegúrate de entender la interacción entre Modelo, Vista y Controlador.

Planificar la estructura: Antes de escribir una línea de código, planifica la estructura de tu aplicación. Define claramente cuáles serán tus Modelos (entidades y lógica de negocio), Vistas (interfaces de usuario) y Controladores (acciones del usuario). Un buen diseño inicial te ahorrará muchos problemas a largo plazo.

Diseñar modelos robustos: Los Modelos son el corazón de tu aplicación MVC. Representan los datos y la lógica de negocio, por lo que deben ser robustos y bien definidos. Encapsula la lógica de negocio dentro de los Modelos para que sean reutilizables y fáciles de mantener.

Implementar controladores eficientes: Los Controladores son los que manejan las acciones del usuario y actualizan los Modelos. Asegúrate de que tus Controladores sean eficientes, validen los datos de entrada y manejen los errores de manera adecuada. Evita poner lógica de negocio compleja dentro de los Controladores; esa debe estar en los Modelos.

## **6.- Conclusiones:**

En resumen, este proyecto ha culminado con la implementación exitosa de un sistema de gestión de inventario que cumple con los requisitos iniciales. La aplicación del patrón MVC ha sido fundamental para lograr una estructura de código organizada y fácil de mantener, facilitando la separación de responsabilidades entre el Modelo, la Vista y el Controlador. A lo largo del desarrollo, se han superado desafíos como la integración de la base de datos y el diseño de una interfaz de usuario intuitiva. El uso de un framework MVC y la realización de pruebas exhaustivas han sido clave para asegurar la calidad del sistema. Este proyecto ha sido una valiosa experiencia de aprendizaje que me ha permitido aplicar mis conocimientos en un contexto real y me motiva a seguir profundizando en el desarrollo de software.

## **7.- Referencias Bibliográficas:**

- Basc, E. (2017). El patrón de diseño Modelo-Vista-Controlador (MVC) y su implementación en Java Swing. Acta Nova, 2(Mvc), 493–507.<https://hdl.handle.net/11042/2743>.
- Salomón, M. S. N. (2021). *Diseño e implementación de un Sistema de control de inventario para Alpasomarket*. <https://dspace.ups.edu.ec/handle/123456789/20931>.
- DispatchTrack,2022. *Sistemas de gestión de inventarios de una empresa: tipos y ejemplos*. (s. f.-b). <https://www.beetrack.com/es/blog/sistemas-de-gestion-de-inventarios-empresa-tipos-y-ejemplos>