# Informe de Funcionalidad y Sustentación de Código: Módulo de Consola PandaTaT

## Proyecto:

Sistema de Gestión de Consola - PandaTaT

## Aprendiz:

Bryan Steep Mora Barrios

## Ficha:

2826487

## Programa:

Análisis y Desarrollo de Software (ADSO)

## 1. Introducción

El propósito de este documento es presentar y sustentar el desarrollo del módulo terminal\_menu.py, un componente integral del proyecto PandaTaT. Este script implementa una interfaz de línea de comandos (CLI) diseñada para gestionar las interacciones del usuario con el sistema, incluyendo la autenticación, el registro y la consulta de información específica según el rol asignado.

El desarrollo se ha fundamentado en la aplicación rigurosa de los paradigmas de la Programación Orientada a Objetos (POO). Se han implementado conceptos clave como Clases, Herencia, Polimorfismo y Decoradores con el fin de construir una solución de software que sea robusta, mantenible y escalable, adhiriéndose a las mejores prácticas de la ingeniería de software.

## 2. Arquitectura del Módulo

La solución de software se compone de tres archivos interdependientes que garantizan su operación integral:

script\_DB.sql: Define el esquema completo de la base de datos PandaTaT en MySQL. Este archivo es responsable de la creación de las tablas (Roles, Estado\_pedidos, Usuarios, Pedidos) y de la inserción de un conjunto de datos inicial (INSERT INTO) para facilitar las pruebas de funcionalidad del sistema.

database.py: Este módulo establece la conexión con el servidor de la base de datos. Su función es encapsular los detalles de la conexión y proporcionar un cursor global (cleverCursor) para que los demás componentes de la aplicación puedan ejecutar consultas SQL de manera centralizada.

terminal\_menu.py: Constituye el núcleo de la lógica de la aplicación. Este archivo orquesta la interfaz de usuario, gestiona el flujo del programa y contiene la implementación detallada de las clases y objetos que modelan el dominio del problema.

## 3. Sustentación Técnica del Código (terminal\_menu.py)

El diseño del código se centra en la aplicación de principios de POO para resolver los requerimientos del sistema de manera estructurada.

## 3.1. Clases y Abstracción de Datos

La entidad central del modelo es la Clase Usuario. Esta clase actúa como una abstracción que encapsula los atributos y comportamientos comunes a todos los usuarios del sistema, tales como id\_usuario, nombre, email y rol.

Se implementaron los métodos login y registrar\_usuario como métodos estáticos (@staticmethod). Esta decisión de diseño se justifica porque dichas operaciones son inherentes a la gestión de usuarios en general y no dependen del estado de una instancia particular. Permiten, por tanto, la autenticación o el registro de un usuario sin necesidad de haber instanciado previamente un objeto de la clase.

## 3.2. Decoradores para la Validación de Datos

Para reforzar la integridad de los datos, se implementó el decorador @validar\_rol.

Funcionalidad: Este decorador envuelve al método guardar\_usuario para interceptar su ejecución. Antes de proceder con la inserción en la base de datos, realiza una consulta para verificar que el rol asignado al nuevo usuario exista en la tabla Roles. Si la validación falla, la operación se aborta.

Justificación de Diseño: El uso de decoradores promueve un código más limpio y modular al separar la lógica de negocio principal de las responsabilidades transversales como la validación. Esto se alinea con el principio DRY (Don't Repeat Yourself) y facilita el mantenimiento futuro del sistema.

## 3.3. Herencia para la Especialización de Roles

El concepto de Herencia se utilizó para modelar la jerarquía de usuarios y reutilizar código de manera efectiva. Se crearon las clases derivadas Cliente, Vendedor y Administrador, las cuales heredan directamente de la clase base Usuario.

Reutilización de Código: Las clases hijas heredan los atributos y métodos de Usuario, evitando la duplicación de código relacionado con la inicialización de atributos y la lógica de autenticación.

Especialización: Cada clase hija extiende la funcionalidad base con métodos específicos para su rol. Por ejemplo, Cliente tiene un método para consultar su historial de pedidos, mientras que Administrador posee métodos para acceder a reportes globales.

## 3.4. Polimorfismo para Flexibilidad Funcional

El Polimorfismo es un pilar fundamental en la arquitectura del menú, manifestado a través del método menu().

Interfaz Común: En la clase base Usuario, el método menu() se declara como una interfaz abstracta (lanzando NotImplementedError). Esto establece un contrato que obliga a todas las clases hijas a proporcionar una implementación concreta de dicho método.

Implementaciones Múltiples: Cada clase hija (Cliente, Vendedor, Administrador) redefine el método menu() para presentar un conjunto de opciones específicas y pertinentes a su rol.

Comportamiento Dinámico: La principal ventaja se observa en la función menu\_principal. Tras una autenticación exitosa, el sistema invoca usuario\_logueado.menu(). En tiempo de ejecución, Python determina dinámicamente qué versión del método menu ejecutar basándose en la clase del objeto usuario\_logueado.

Este enfoque polimórfico dota al sistema de una gran flexibilidad. La adición de nuevos roles en el futuro solo requeriría la creación de una nueva clase que herede de Usuario e implemente su propio método menu, sin necesidad de modificar el flujo principal del programa.

## 4. Conclusión

El módulo terminal\_menu.py implementa de manera satisfactoria la funcionalidad requerida, ofreciendo una interfaz de consola estructurada y segura para los usuarios del sistema PandaTaT.