# 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 presente informe detalla el desarrollo y la implementación de una **API RESTful** para el proyecto **PandaTaT**. El objetivo de esta actividad fue construir el backend de un sistema capaz de gestionar usuarios, pedidos, roles y estados, aplicando principios de arquitectura de software moderna y paradigmas de la Programación Orientada a Objetos (POO).

La solución final es un servicio web funcional que expone una serie de endpoints para ser consumidos por una aplicación cliente (frontend), cumpliendo con todos los requisitos especificados en la actividad.

## 2. Arquitectura del Sistema (Modelo-Vista-Controlador)

El proyecto se estructuró siguiendo una adaptación del patrón arquitectónico **Modelo-Vista-Controlador (MVC)**, ideal para el desarrollo de APIs web:

* **Modelo**: Representa la capa de datos y la lógica de negocio. En este proyecto, se compone de:
  + **Base de Datos (script\_DB.sql)**: Se creó una base de datos en MySQL llamada PandaTaT con las tablas Usuarios, Pedidos, Roles y Estado\_pedidos para asegurar la persistencia y la integridad de la información.
  + **Conector (database.py)**: Módulo que establece y gestiona la conexión con la base de datos MySQL utilizando la dependencia mysql.connector.
  + **Modelos de Datos (models.py)**: Clases de Pydantic que definen la estructura de los datos dentro de la aplicación (ej. Usuario, Pedido), garantizando la validación y consistencia de la información.
* **Vista**: Es la representación de los datos que se entregan al cliente.
  + **Respuestas JSON**: La API no genera una vista visual directa. En su lugar, los endpoints devuelven los datos en formato **JSON (JavaScript Object Notation)**. Este formato es el estándar para la comunicación entre un backend y un frontend, actuando como la "vista" que el cliente interpretará.
* **Controlador**: Recibe las peticiones del cliente (frontend), interactúa con el **Modelo** para procesar la solicitud y devuelve la **Vista** (JSON) correspondiente.
  + **Endpoints (main.py)**: Este archivo, utilizando el framework FastAPI, contiene las funciones que definen los diferentes endpoints de la API (ej. /login, /pedidos, /usuarios/{rol}). Cada una de estas funciones actúa como un controlador que gestiona una ruta específica.

## 3. Aplicación de Conceptos de Programación Orientada a Objetos

Los principios de POO fueron fundamentales para lograr un código modular, reutilizable y fácil de mantener.

* **Clases y Métodos**: Se definieron clases como UsuarioBase, Usuario y Pedido en el archivo models.py. Estas clases no solo estructuran los datos, sino que encapsulan la información relevante de cada entidad del sistema. FastAPI utiliza los métodos internos de estas clases (heredados de Pydantic BaseModel) para realizar la validación y serialización de datos.
* **Herencia**: El concepto de herencia se aplicó en models.py, donde la clase Usuario hereda de la clase UsuarioBase. Esto permite reutilizar los campos comunes (nombre, apellido, email) y añadir atributos específicos en la clase hija, evitando la duplicación de código.
* **Decoradores**: Los decoradores son una característica clave del framework FastAPI. Cada endpoint se define con un decorador que especifica la ruta y el método HTTP (ej. @app.post("/login")). Esto demuestra cómo los decoradores pueden añadir funcionalidades (en este caso, el enrutamiento web) a una función de Python de manera limpia y declarativa.
* **Polimorfismo**: Aunque es más sutil en una API, el polimorfismo se manifiesta en la capacidad del framework para manejar diferentes objetos de modelo (LoginData, Pedido, etc.) a través de una interfaz común. El sistema procesa diferentes estructuras de datos de manera consistente, validando y serializando la información según la clase del objeto recibido o devuelto.

## 4. Cumplimiento de Requisitos de la Actividad

Se han cumplido todos los puntos solicitados en la actividad:

1. **Base de Datos "PandaTaT"**: Creada y estructurada con las tablas solicitadas mediante el script script\_DB.sql.
2. **Estructura MVC**: Implementada de forma adaptada para una API RESTful.
3. **Conceptos POO**: Aplicados en la definición de modelos y en la estructura del código.
4. **Conexión a BD**: Realizada exitosamente con la dependencia mysql.connector en database.py.
5. **Endpoints**: Se crearon endpoints funcionales para cada tipo de consulta requerida, como la autenticación de usuarios, la consulta general de pedidos y el filtrado de usuarios por rol.
6. **Proyecto Funcional y Versionado**: El proyecto es un entregable funcional y está preparado para ser versionado con **Git**, incluyendo un archivo .gitignore para las buenas prácticas de control de versiones.

## 5. Conclusión

El proyecto final es una API RESTful robusta y bien estructurada que sirve como una base sólida para el desarrollo de una aplicación completa. La utilización de FastAPI y la aplicación de los principios de MVC y POO han permitido crear un backend eficiente, escalable y fácil de mantener, cumpliendo con todos los objetivos pedagógicos y técnicos de la actividad.