**Pruebas y depuración de programas**

**Semana 8**

Sara C. Buriticá Olaya

Camilo S. Cordoba Russi

Corporación Universitaria Minuto de Dios

Facultad de ingeniería, Esp. Desarrollo de Software, Plataformas de Desarrollo de Software

Mg. Ricardo A. Fonseca Perdomo

1 de septiembre de 2024

**Introducción**

En las semanas previas, se comenzó el desarrollo de una solución de software para la Comercializadora GREMLINS, abordando inicialmente un análisis detallado de los requisitos funcionales y no funcionales de la compañía. Este análisis permitió definir con precisión los objetivos del proyecto y establecer una base sólida para su implementación.

Sobre esta base, se amplió y perfeccionó el trabajo mediante la implementación de pruebas y la depuración del software desarrollado. Este proceso fue fundamental para asegurar que la solución no solo cumpliera con los objetivos planteados, sino que también operara de manera eficiente y sin errores.

El presente documento continúa con la exploración y desarrollo de esta solución, utilizando Python como lenguaje de programación, la librería tkinter para la creación de interfaces gráficas, y Visual Studio Code como editor de código.

### Preguntas orientadoras

### ¿Por qué es importante realizar pruebas a los programas?

La importancia de realizar pruebas radica en que las pruebas permiten verificar que el software funcione conforme a los requisitos y expectativas de los usuarios. Las pruebas son fundamentales para identificar errores y defectos antes de que el software se libere al usuario final.

Aquí están los puntos clave:

* **Validación de Requisitos:** Las pruebas aseguran que el software cumple con los requisitos especificados en el diseño. Validan que todas las funcionalidades se implementen correctamente y que el software realice las tareas esperadas.
* **Detección Temprana de Defectos:** Al realizar pruebas en diferentes etapas del desarrollo, se pueden detectar defectos antes de que el software llegue a producción. Esto ayuda a evitar problemas que podrían afectar la experiencia del usuario o la funcionalidad del sistema.
* **Mejora de la Calidad:** Las pruebas contribuyen a mejorar la calidad general del software al identificar y corregir errores, inconsistencias y problemas de rendimiento. Esto resulta en un producto final más robusto y confiable.
* **Garantía de Funcionamiento:** Aseguran que el software funcione de manera estable y eficaz en diversas condiciones y escenarios de uso. Esto incluye verificar la usabilidad, la seguridad y la compatibilidad del software.
* **Satisfacción del Usuario:** La realización de pruebas ayuda a garantizar que el software cumpla con las expectativas de los usuarios finales, lo cual es crucial para la aceptación y el éxito del producto en el mercado.

**¿Por qué es importante realizar depuración a los programas?**

La depuración consiste en identificar, analizar y corregir errores o "bugs" en el código. Aunque a menudo se confunde con las pruebas, la depuración tiene un enfoque más profundo y detallado, centrándose en el examen y la corrección de problemas específicos que pueden afectar la funcionalidad y el rendimiento del software. Aquí están los puntos clave:

* **Resolución de Problemas Específicos:** La depuración permite localizar el origen de errores específicos que se han identificado durante las pruebas o el uso del software. Esto es crucial para solucionar problemas que afectan el funcionamiento del software.
* **Análisis Detallado del Código:** Permite un análisis profundo del código fuente para entender y corregir errores. La depuración ayuda a encontrar fallos que pueden no ser evidentes a través de pruebas generales.
* **Mejora de la Estabilidad:** Al corregir errores en el código, la depuración mejora la estabilidad y el rendimiento del software. Esto asegura que el software funcione de manera confiable en diferentes situaciones y escenarios de uso.
* **Prevención de Problemas Futuros:** La depuración no solo corrige errores actuales, sino que también ayuda a prevenir problemas futuros al mejorar la calidad del código. Un código bien depurado es menos propenso a introducir errores adicionales.
* **Optimización del Desarrollo:** Facilita la solución de problemas y reduce el tiempo necesario para resolver defectos específicos. Esto acelera el proceso de desarrollo y mejora la eficiencia general del equipo de desarrollo.

### Estudio de Caso: Optimización del Proceso Logístico en Comercializadora Gremlins

**Link del desarrollo:** <https://github.com/saraburiticao/Comercializadora_Gremlins>

**Resumen del trabajo presentado en la Semana 7.**

**Aplicación Propuesta: Gremlins Logistics**

* **Objetivo Principal:** Optimizar el proceso logístico de importación, gestión de distribuidores y compradores, reempaque de productos, seguimiento de envíos y mantenimiento de información, con el fin de aumentar la demanda y mejorar la eficiencia operativa de la comercializadora en Colombia.

### Actores:

1. **Gerente:**
   * Responsable de la gestión de importaciones y el control de proveedores.
   * Toma decisiones estratégicas sobre la clasificación de productos y la ocultación de información de proveedores para asegurar la ventaja competitiva.
2. **Administrador:**
   * Encargado de la gestión de datos tanto de distribuidores como de compradores.
   * Administra los inventarios y genera reportes esenciales para la toma de decisiones y el análisis de desempeño.
3. **Empacador:**
   * Maneja el reempaque y etiquetado de los productos según las especificaciones del comprador.
   * Asegura que los productos estén correctamente preparados para su distribución.
4. **Transportador:**
   * Responsable de la entrega y seguimiento de los paquetes a los clientes finales.
   * Gestiona rutas y horarios de entrega, y permite la actualización de datos relevantes para la distribución.
5. **Recepcionista:**
   * Mantiene actualizada la información de clientes y distribuidores.
   * Facilita la comunicación interna y asegura que todos los datos relevantes estén disponibles para el correcto proceso de transporte y entrega.

### Contexto:

La aplicación está diseñada para una comercializadora internacional llamada Gremlins, que importa productos desde el exterior hacia Colombia. Gremlins se enfrenta al reto de optimizar su proceso logístico para mejorar la eficiencia operativa y fortalecer su posición en el mercado. Los productos importados necesitan ser clasificados y reempaquetados, ocultando la información de los proveedores para mantener la confidencialidad y la ventaja competitiva de la empresa.

Cada actor en la aplicación tiene un rol clave que contribuye al proceso global, desde la gestión de importaciones hasta la entrega final de los productos. La solución propuesta busca integrar todas estas funciones en una plataforma unificada que permita una mejor coordinación, trazabilidad y manejo de la información, con el objetivo de satisfacer las necesidades modernas de los clientes y asegurar el crecimiento sostenible de la empresa en un entorno de negocios altamente competitivo.

**Selección de Roles:**

Debajo del mensaje de bienvenida, la aplicación solicita al usuario que **seleccione su rol**. Los roles disponibles son:

* Gerente
* Administrador
* Empacador
* Transportador
* Recepción

Cada uno de estos roles está representado por un botón en la interfaz. Al hacer clic en uno de estos botones, la aplicación reconoce qué rol ha sido seleccionado y muestra un mensaje en la consola (en este caso, sólo para fines de prueba). Este es un punto de partida; en una versión más avanzada de la aplicación, cada rol podría tener funcionalidades específicas que se activan al seleccionarlo.

**Código Desarrollado**

### Paso 1: Instalación y configuración inicial

1. **Instalar Python:**

Se descarga desde [python.org](https://www.python.org/).

1. **Configurar el entorno de desarrollo:**

Se utiliza un editor de código como Visual Studio Code para la elaboración del proyecto.

1. **Crear un nuevo archivo Python:**

Se abre el editor de código y se crean dos archivos:

“comercializadora\_gremlins\_app.py”

“comercializadora\_gremlins\_modules.py”

**Paso 2: Crear la página de la aplicación - “comercializadora\_gremlins\_app.py”**

**2.1. Importar las bibliotecas necesarias:** Estas líneas importan tkinter y su módulo de fuentes para crear la interfaz gráfica.

import tkinter as tk

from tkinter import font

**2.2. Definir funciones para la interacción del usuario:** La función on\_role\_selected(role) maneja la selección del rol, mientras que on\_exit() cerrará la aplicación.

def on\_role\_selected(role):

print(f"Rol seleccionado: {role}")

def on\_exit():

root.destroy()

**2.3. Crear la ventana principal de la aplicación:** Esto configura la ventana principal con un título y tamaño específicos.

**root = tk.Tk()**

**root.title("Comercializadora Gremlins")**

**root.geometry("600x400")**

**2.4. Configurar el título principal:** Aquí se crea un marco (frame) y un título para la aplicación.

# Configuración del título principal

frame = tk.Frame(root, padx=20, pady=20, relief="solid", bd=2)

frame.pack(expand=True)

titulo\_font = font.Font(family="Helvetica", size=16, weight="bold")

titulo = tk.Label(frame, text="COMERCIALIZADORA GREMLINS", font=titulo\_font)

titulo.pack(pady=10)

**2.5. Añadir un mensaje de bienvenida y selección de rol:** Se agregan mensajes de bienvenida y una etiqueta para la selección del rol.

**bienvenido\_font = font.Font(family="Helvetica", size=12, weight="bold")**

**bienvenido = tk.Label(frame, text="Bienvenidos", font=bienvenido\_font)**

**bienvenido.pack(pady=10)**

**# Configuración del mensaje de selección de rol**

**seleccion\_font = font.Font(family="Helvetica", size=10, weight="bold")**

**seleccion\_label = tk.Label(frame, text="Seleccione su rol.", font=seleccion\_font)**

**seleccion\_label.pack(pady=10)**

**2.6. Crear botones para cada rol :** Aquí se crean botones que representan los diferentes roles disponibles en la aplicación.

**# Configuración de los botones de rol**

**roles = ["Gerente", "Administrador", "Empacador", "Transportador", "Recepción"]**

**button\_frame = tk.Frame(frame)**

**button\_frame.pack(pady=10)**

**for rol in roles:**

**button = tk.Button(button\_frame, text=rol, command=lambda r=rol: on\_role\_selected(r), padx=10, pady=5)**

**button.pack(side=tk.LEFT, padx=5)**

**2.7. Botón salir y ejecución de la app** : Se agrega un botón para cerrar la aplicación, y una última línea que inicia el ciclo de eventos de Tkinter, mostrando la ventana principal.

**# Botón de salir**

**salir\_button = tk.Button(frame, text="Salir", command=on\_exit, padx=10, pady=5)**

**salir\_button.pack(pady=20)**

**root.mainloop()**

**Paso 3. Creación de los modulos -** “comercializadora\_gremlins\_modules.py”

**3.1. Importar las bibliotecas necesarias**

**import tkinter as tk**

**from tkinter import font, messagebox**

**3.2. Definir funciones para abrir diferentes módulos:** Estas funciones abren diferentes módulos de la aplicación, cada uno relacionado con un rol específico.

**# Funciones para abrir diferentes módulos basados en el rol**

**def open\_import\_management():**

**create\_module\_window("Gestión de Importaciones - Gerente", ["Clasificación y gestión de productos importados", "Asignación y gestión de proveedores ocultos", "Generación de informes"])**

**def open\_admin\_management():**

**create\_module\_window("Módulo de Administración", ["Gestión de usuarios", "Administración de inventarios", "Generación de reportes"])**

**def open\_packing\_management():**

**create\_module\_window("Módulo de Empaque", ["Empaque de productos", "Revisión de productos empacados", "Actualización de inventario"])**

**def open\_transport\_management():**

**create\_module\_window("Módulo de Transporte", ["Gestión de rutas", "Monitoreo de entregas", "Registro de transportes"])**

**def open\_reception\_management():**

**create\_module\_window("Módulo de Recepción", ["Recepción de productos", "Clasificación inicial", "Registro de recibos"])**

**3.3. Crear una función genérica para las ventanas de módulos:** Esta función crea una ventana de módulo con un título y opciones basadas en el rol seleccionado.

# Función genérica para crear una ventana de módulo

def create\_module\_window(title, options):

module\_window = tk.Toplevel(root)

module\_window.title(title)

module\_window.geometry("600x400")

title\_font = font.Font(family="Helvetica", size=14, weight="bold")

title\_label = tk.Label(module\_window, text=title, font=title\_font)

title\_label.pack(pady=10)

for option in options:

button = tk.Button(module\_window, text=option, padx=20, pady=10)

button.pack(pady=10)

close\_button = tk.Button(module\_window, text="Cerrar", command=module\_window.destroy, padx=20, pady=10)

close\_button.pack(pady=20)

**3.4. Selección del rol y acceso a cada módulo:** Dependiendo del rol seleccionado, se abrirá el módulo correspondiente.

**# Selección del rol y acceso al módulo correspondiente**

**def on\_role\_selected(role):**

**if role == "Gerente":**

**open\_import\_management()**

**elif role == "Administrador":**

**open\_admin\_management()**

**elif role == "Empacador":**

**open\_packing\_management()**

**elif role == "Transportador":**

**open\_transport\_management()**

**elif role == "Recepción":**

**open\_reception\_management()**

**else:**

**messagebox.showinfo("Acceso Restringido", f"El rol '{role}' no tiene acceso a ningún módulo.")**

**Reutilizar la ventana principal:**

● Se reutiliza el código de la primera página para la configuración de la ventana principal, añadiendo los botones que llamarán a las funciones de los módulos.

● Finalmente, como en el primer archivo, utiliza root.mainloop() para ejecutar la aplicación.

# Función para salir de la aplicación

def on\_exit():

root.destroy()

# Configuración de la ventana principal

root = tk.Tk()

root.title("Comercializadora Gremlins")

root.geometry("600x400")

# Configuración del título principal

frame = tk.Frame(root, padx=20, pady=20, relief="solid", bd=2)

frame.pack(expand=True)

titulo\_font = font.Font(family="Helvetica", size=16, weight="bold")

titulo = tk.Label(frame, text="COMERCIALIZADORA GREMLINS", font=titulo\_font)

titulo.pack(pady=10)

bienvenido\_font = font.Font(family="Helvetica", size=12, weight="bold")

bienvenido = tk.Label(frame, text="Bienvenidos", font=bienvenido\_font)

bienvenido.pack(pady=10)

# Configuración del mensaje de selección de rol

seleccion\_font = font.Font(family="Helvetica", size=10, weight="bold")

seleccion\_label = tk.Label(frame, text="Seleccione su rol.", font=seleccion\_font)

seleccion\_label.pack(pady=10)

# Configuración de los botones de rol

roles = ["Gerente", "Administrador", "Empacador", "Transportador", "Recepción"]

button\_frame = tk.Frame(frame)

button\_frame.pack(pady=10)

for rol in roles:

button = tk.Button(button\_frame, text=rol, command=lambda r=rol: on\_role\_selected(r), padx=10, pady=5)

button.pack(side=tk.LEFT, padx=5)

# Botón de salir

salir\_button = tk.Button(frame, text="Salir", command=on\_exit, padx=10, pady=5)

salir\_button.pack(pady=20)

root.mainloop()

### Funcionalidades de la Aplicación:

La aplicación se enfoca en los requisitos tomados por la Comercializadora Gremlins seleccionado un rol en específico y solucionando el problema de cada área correspondiente, se desarrolla una aplicación de fácil uso e intuitiva incluyendo su manual de usuario y en caso de tener dudas un correo de soporte.

Cada área en específico cuenta con un módulo en específico y 3 submódulos que definen cada acción a realizar. Finalmente, la aplicación incluye un botón de **Salir** en la parte inferior, que permite a los usuarios cerrar la ventana de manera segura cuando terminan de usar la aplicación.

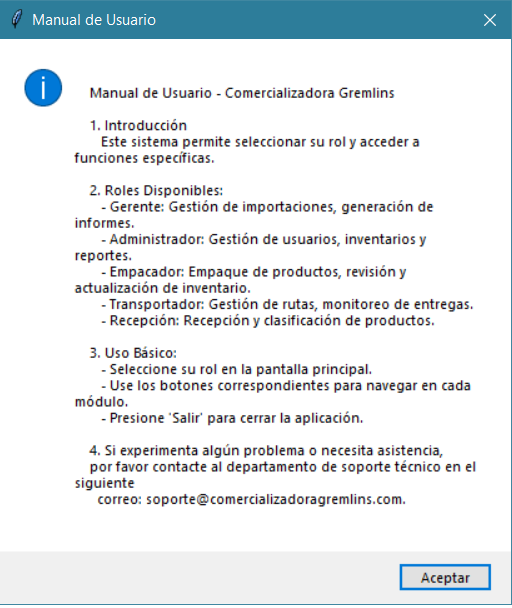
**Interfaces gráficas**

### Imagen 7. Interfaz Gráfica de Inicio

### 

**Fuente:** Imagen propia

**Imagen 8.** Manual de usuario y botón salir > Interfaz de bienvenida

****

**Fuente:** Imagen propia

**Imagen 9.** Módulo Gerente

### 

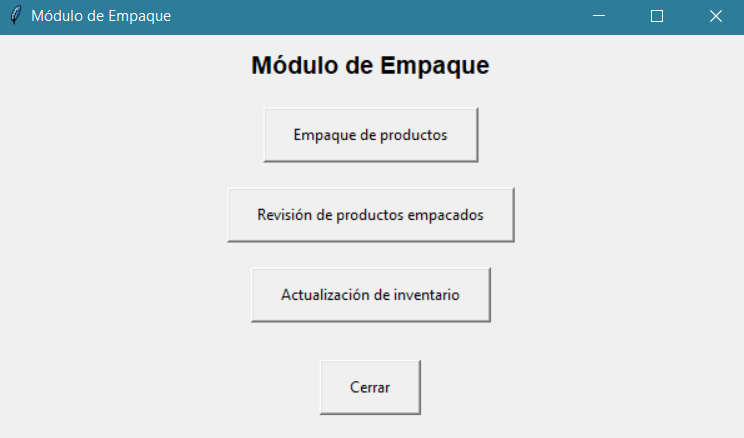
**Fuente:** Imagen propia

**Imagen 10.** Módulo Administración



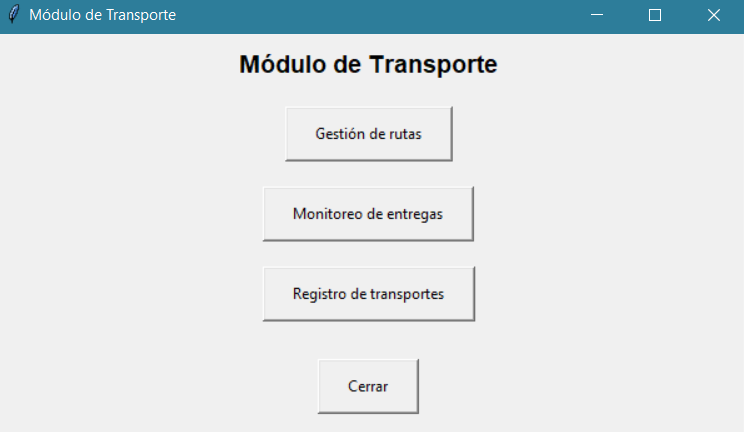
**Fuente:** Imagen propia

**Imagen 11.** Modulo Empacador



**Fuente:** Imagen propia

**Imagen 12. Módulo** Transportador



**Fuente:** Imagen propia

**Imagen 13.**  Módulo Recepción



**Fuente:** Imagen propia

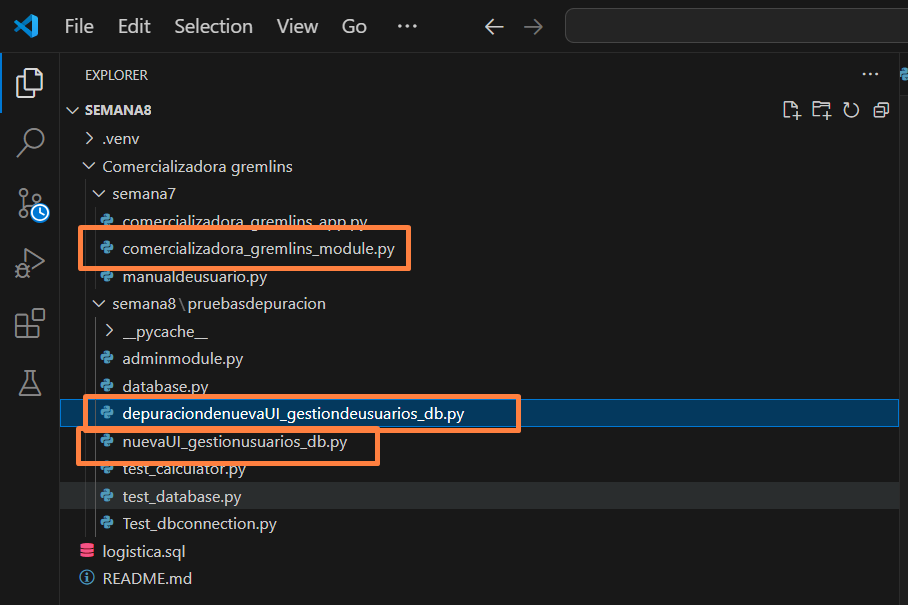
**Semana 8**

**Pruebas y depuración del programa**

En la semana 8, se complementó el proyecto desarrollado en la semana 7. Se retomó el software y se conectó el proyecto a una base de datos en el módulo de administración, con el objetivo de visualizar los usuarios que tienen acceso a la aplicación y realizar pruebas. Para estos fines, el código de la aplicación en la semana 8 presenta algunos aciertos y errores, los cuales fueron depurados posteriormente.

* Para la semana 7. la interfaz de usuario inicial se puede observar en el archivo comercializadora\_gremlins\_module.py.
* Para la semana 8. 7 la interfaz de usuario con errores se puede observar en el archivo nuevaUI\_gestionusuarios\_db.py
* la interfaz de usuario con la depuración aplicada se puede observar en el archivo depuraciondenuevaUI\_gestiondeusuarios\_db.py

**Imagen 14.** Archivos en visual code



**Fuente:** propia

Paso 1. Cambios implementados para el código “nuevaUI\_gestionusuarios\_db.py” se hicieron partiendo de una comparativa de los Módulos de Gestión:

**Semana 7.**

* Está más generalizado y se crean ventanas para diferentes módulos, pero sin funcionalidades específicas implementadas.
* La función create\_module\_window es más simple y no incluye ningún callback. Se limita a crear una ventana con botones para las opciones que se le pasan.
* No incluye un Treeview ni ningún otro widget para mostrar datos de forma tabular. Es más básico en términos de interfaz y funcionalidades.

**Semana 8.**

* Incluye un módulo específico para la gestión de usuarios con una función llamada open\_user\_management, que abre una ventana con un Treeview para mostrar una lista de usuarios obtenidos desde la base de datos.
* La función create\_module\_window recibe un callback opcional (open\_user\_management\_callback), lo que permite personalizar la acción de un botón específico (como el de gestión de usuarios). Esto agrega flexibilidad, pero también hace que la función sea más compleja.
* Al seleccionar un rol, se llama a la función on\_role\_selected, que invoca directamente la función correspondiente al rol elegido (como open\_admin\_management o open\_packing\_management).
* Incluye un Treeview para mostrar una tabla de usuarios dentro de la ventana de gestión de usuarios, lo que sugiere un enfoque más avanzado y específico en la gestión de datos.

Código de nuevaUI\_gestionusuarios\_db.py con las mejoras comentadas anteriormente.

import tkinter as tk

from tkinter import font, ttk

from database import get\_users

# Funciones para abrir las ventanas de gestión

def open\_user\_management():

    user\_window = tk.Toplevel(root)

    user\_window.title("Gestión de Usuarios")

    user\_window.geometry("600x400")

    title\_font = font.Font(family="Helvetica", size=14, weight="bold")

    title\_label = tk.Label(user\_window, text="Gestión de Usuarios", font=title\_font)

    title\_label.pack(pady=10)

    # Crear un árbol (Treeview) para mostrar los datos

    tree = ttk.Treeview(user\_window, columns=("ID", "Nombre", "Cargo"), show="headings")

    tree.heading("ID", text="ID")

    tree.heading("Nombre", text="Nombre")

    tree.heading("Cargo", text="Cargo")

    tree.pack(expand=True, fill="both")

    # Obtener los datos de la base de datos

    users = get\_users()

    for user in users:

        tree.insert("", tk.END, values=user)

    close\_button = tk.Button(user\_window, text="Cerrar", command=user\_window.destroy, padx=20, pady=10)

    close\_button.pack(pady=20)

def open\_inventory\_management():

    # Implementa la lógica para la gestión de inventarios aquí

    pass

def open\_reports\_management():

    # Implementa la lógica para la generación de reportes aquí

    pass

def open\_packing\_management():

    # Implementa la lógica para la gestión de empaques aquí

    pass

def open\_transport\_management():

    # Implementa la lógica para la gestión de transporte aquí

    pass

def open\_reception\_management():

    # Implementa la lógica para la gestión de recepción aquí

    pass

# Función para crear la ventana del módulo de administración

def open\_admin\_management():

    create\_module\_window(

        title="Módulo de Administración",

        options=["Gestión de usuarios", "Administración de inventarios", "Generación de reportes"],

        open\_user\_management\_callback=open\_user\_management

    )

# Función genérica para crear una ventana de módulo

def create\_module\_window(title, options, open\_user\_management\_callback=None):

    module\_window = tk.Toplevel(root)

    module\_window.title(title)

    module\_window.geometry("600x400")

    title\_font = font.Font(family="Helvetica", size=14, weight="bold")

    title\_label = tk.Label(module\_window, text=title, font=title\_font)

    title\_label.pack(pady=10)

    for option in options:

        if option == "Gestión de usuarios" and open\_user\_management\_callback:

            button = tk.Button(module\_window, text=option, padx=20, pady=10, command=open\_user\_management\_callback)

        elif option == "Administración de inventarios":

            button = tk.Button(module\_window, text=option, padx=20, pady=10, command=open\_inventory\_management)

        elif option == "Generación de reportes":

            button = tk.Button(module\_window, text=option, padx=20, pady=10, command=open\_reports\_management)

        elif option == "Gestión de empaques":

            button = tk.Button(module\_window, text=option, padx=20, pady=10, command=open\_packing\_management)

        elif option == "Gestión de transporte":

            button = tk.Button(module\_window, text=option, padx=20, pady=10, command=open\_transport\_management)

        elif option == "Gestión de recepción":

            button = tk.Button(module\_window, text=option, padx=20, pady=10, command=open\_reception\_management)

        else:

            button = tk.Button(module\_window, text=option, padx=20, pady=10)

        button.pack(pady=10)

    close\_button = tk.Button(module\_window, text="Cerrar", command=module\_window.destroy, padx=20, pady=10)

    close\_button.pack(pady=20)

# Función para manejar la selección de roles

def on\_role\_selected(role):

    if role == "Gerente":

        open\_admin\_management()

    elif role == "Administrador":

        open\_admin\_management()

    elif role == "Empacador":

        open\_packing\_management()

    elif role == "Transportador":

        open\_transport\_management()

    elif role == "Recepción":

        open\_reception\_management()

    else:

        print("Rol desconocido")

# Función para salir de la aplicación

def on\_exit():

    root.destroy()

# Configuración de la ventana principal

root = tk.Tk()

root.title("Comercializadora Gremlins")

root.geometry("600x400")

# Configuración del título principal

frame = tk.Frame(root, padx=20, pady=20, relief="solid", bd=2)

frame.pack(expand=True)

titulo\_font = font.Font(family="Helvetica", size=16, weight="bold")

titulo = tk.Label(frame, text="COMERCIALIZADORA GREMLINS", font=titulo\_font)

titulo.pack(pady=10)

bienvenido\_font = font.Font(family="Helvetica", size=12, weight="bold")

bienvenido = tk.Label(frame, text="Bienvenidos", font=bienvenido\_font)

bienvenido.pack(pady=10)

# Configuración del mensaje de selección de rol

seleccion\_font = font.Font(family="Helvetica", size=10, weight="bold")

seleccion\_label = tk.Label(frame, text="Seleccione su rol.", font=seleccion\_font)

seleccion\_label.pack(pady=10)

# Configuración de los botones de rol

roles = ["Gerente", "Administrador", "Empacador", "Transportador", "Recepción"]

button\_frame = tk.Frame(frame)

button\_frame.pack(pady=10)

for rol in roles:

    button = tk.Button(button\_frame, text=rol, command=lambda r=rol: on\_role\_selected(r), padx=10, pady=5)

    button.pack(side=tk.LEFT, padx=5)

# Botón de salir

salir\_button = tk.Button(frame, text="Salir", command=on\_exit, padx=10, pady=5)

salir\_button.pack(pady=20)

root.mainloop()

luego de desarrollada la nueva interfaz de usuario se procede a realizar las pruebas al programa.

**Pruebas del programa**

**Prueba Unitaria** - **verificar la función responsable de establecer la conexión a la DB**

Se crea el archivo test\_database.py, en el cual se ejecuta una prueba para la función responsable de establecer la conexión, aislándola de otras partes del código.

import unittest

import mysql.connector

def connect\_to\_db():

    connection = mysql.connector.connect(

        host='localhost',

        user='root',

        password='root',

        database='logistica\_db'

    )

    return connection

class TestDatabaseConnection(unittest.TestCase):

    def test\_connection(self):

        connection = connect\_to\_db()

        self.assertIsNotNone(connection)

        self.assertTrue(connection.is\_connected())

        connection.close()

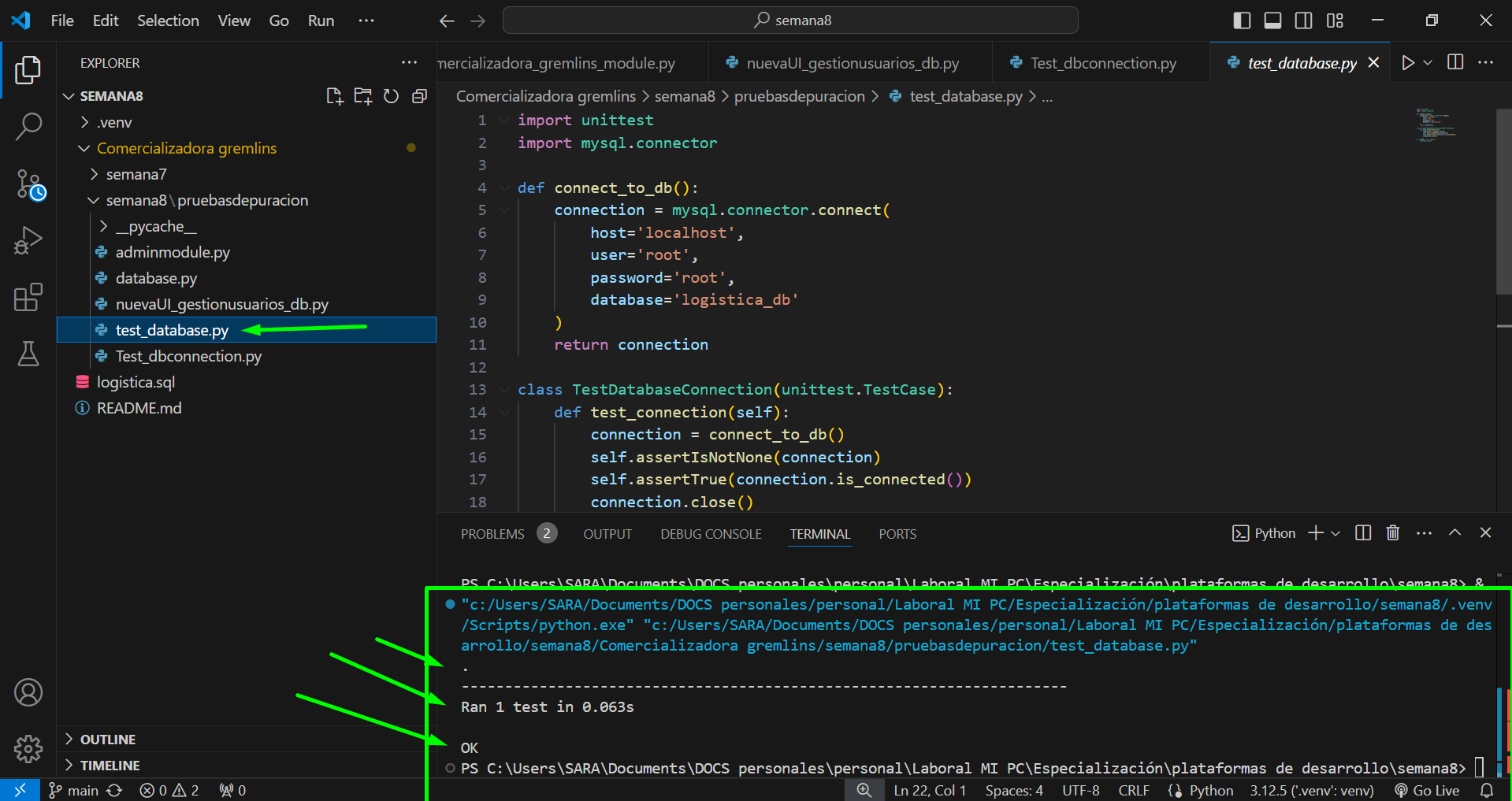
if \_\_name\_\_ == '\_\_main\_\_':

    unittest.main()

En los Resultados obtenidos luego de ejecutar el script test\_database.py se enuentra que:

* El punto. indica que una prueba fue ejecutada y pasó exitosamente.
* “Ran 1 test in 0.063 s” significa que se ejecutó una prueba y tomó 0.063 segundos en completarse.
* El “OK” Significa que la prueba pasó sin ningún problema.

**Imagen 15.** Resultado unitaria para la conexión a la DB



**Fuente:** propia

**Prueba Unitaria - Consulta a la Base de Datos**

Se crea el archivo test\_calculator.py está diseñado para verificar la correcta ejecución de una función que se conecta a la base de datos y recupera todos los usuarios. La prueba comprueba que la lista de usuarios obtenida desde la base de datos coincida con una lista esperada de usuarios.

import unittest

import mysql.connector

# Función para obtener todos los usuarios desde la base de datos

def get\_all\_users():

    conn = mysql.connector.connect(

        host="localhost",

        user="root",

        password="root",

        database="logistica\_db"

    )

    cursor = conn.cursor()

    cursor.execute("SELECT \* FROM usuarios")

    users = cursor.fetchall()

    conn.close()

    return users

class TestDatabaseQueries(unittest.TestCase):

    def test\_get\_all\_users(self):

        # Ejecuta la consulta

        users = get\_all\_users()

        # Define el resultado esperado

        expected\_users = [

            (1, 'Camilo Cordoba', 'Gerente'),

            (2, 'Carlos Díaz', 'Recepción'),

            (3, 'Sara Buriticá', 'Administrador'),

            (4, 'Laura Rivas', 'Transportista'),

            (5, 'Carolina Molina', 'Asistente de Gerente'),

            (6, 'Astrid Pineda', 'Atención al Cliente'),

            (7, 'Estefanía Olaya', 'Transportista'),

            (8, 'Daniela Ramírez', 'Transportista')

        ]

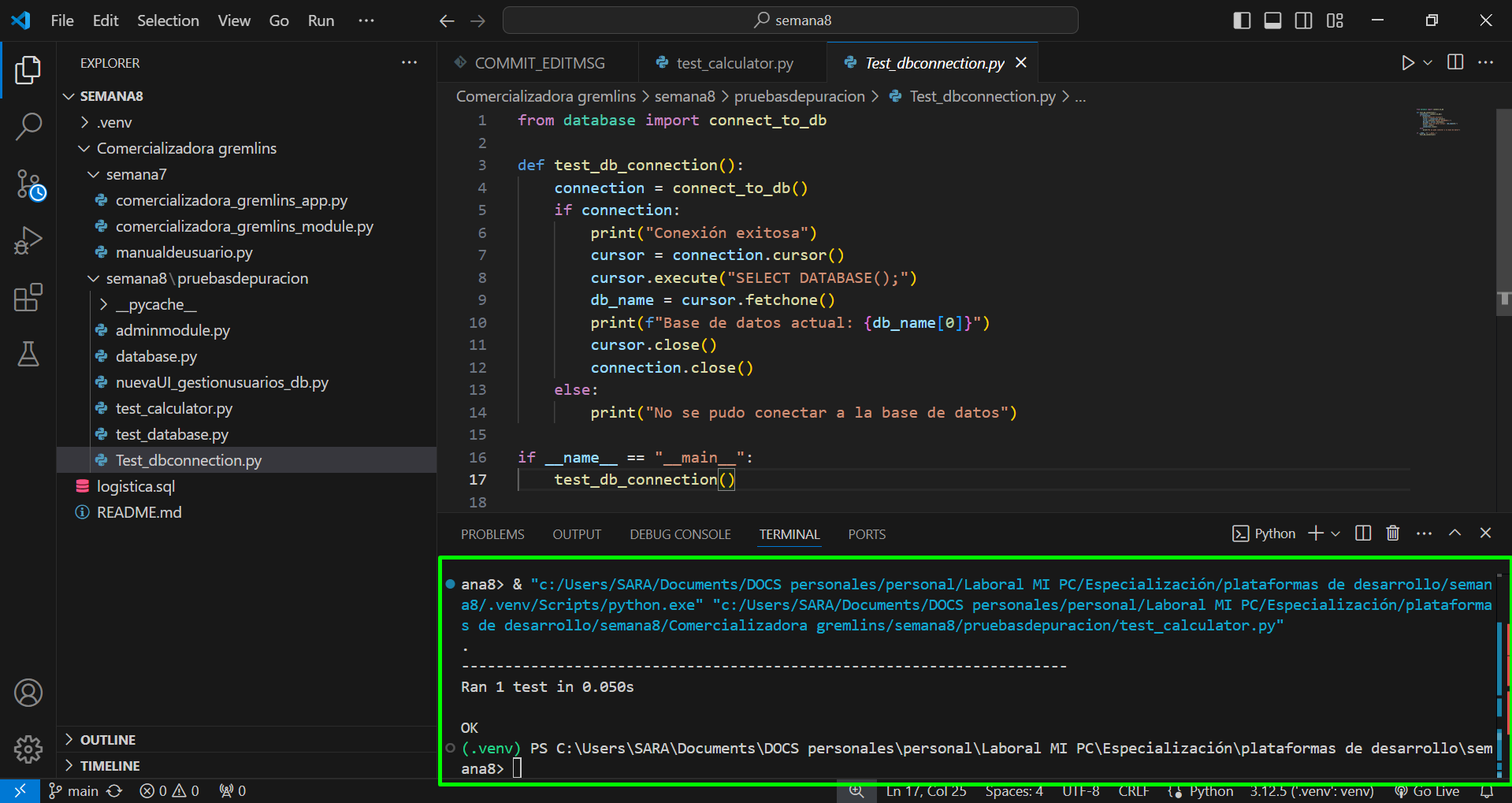
        # Verifica que los resultados sean los esperados

        self.assertEqual(users, expected\_users)

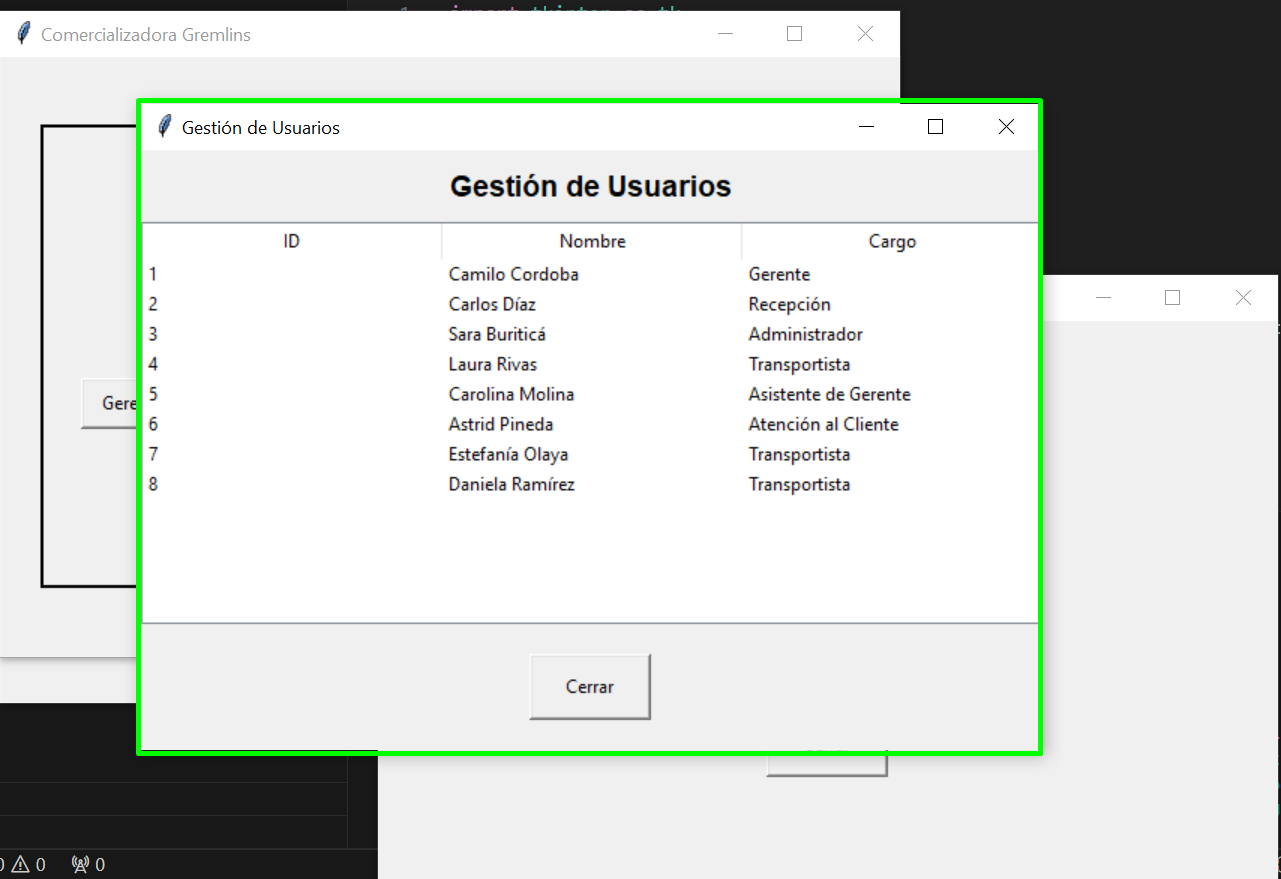
if \_\_name\_\_ == '\_\_main\_\_':

    unittest.main()

**Imagen 16.** Resultado Compatibilidad con la Base de Datos

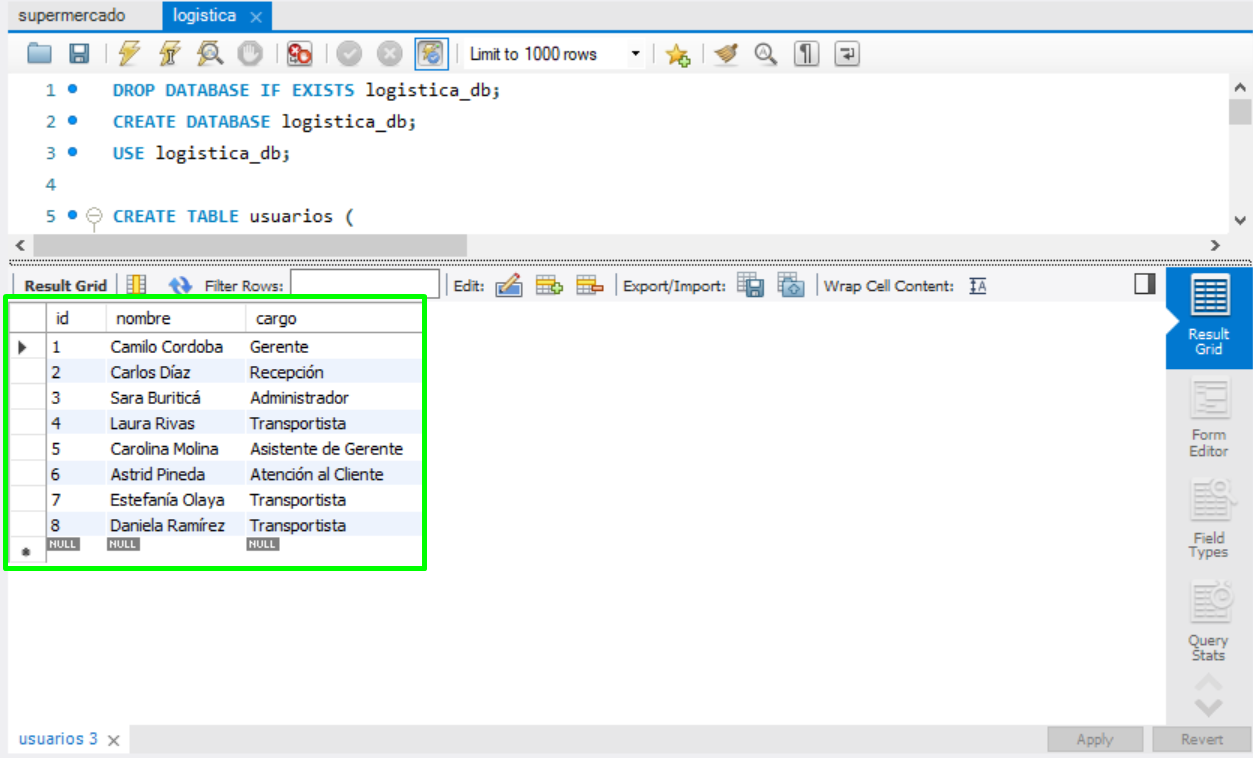
**Fuente:** propia

**Imagen 17.** Interfaz gráfica



**Fuente**: propia

**Imagen 18.** Base de datos



**Fuente :** propia

**Prueba de integración**

Se realizó para verificar que la integración entre la aplicación Python y el sistema de base de datos funciona correctamente.

from database import connect\_to\_db

def test\_db\_connection():

    connection = connect\_to\_db()

    if connection:

        print("Conexión exitosa")

        cursor = connection.cursor()

        cursor.execute("SELECT DATABASE();")

        db\_name = cursor.fetchone()

        print(f"Base de datos actual: {db\_name[0]}")

        cursor.close()

        connection.close()

    else:

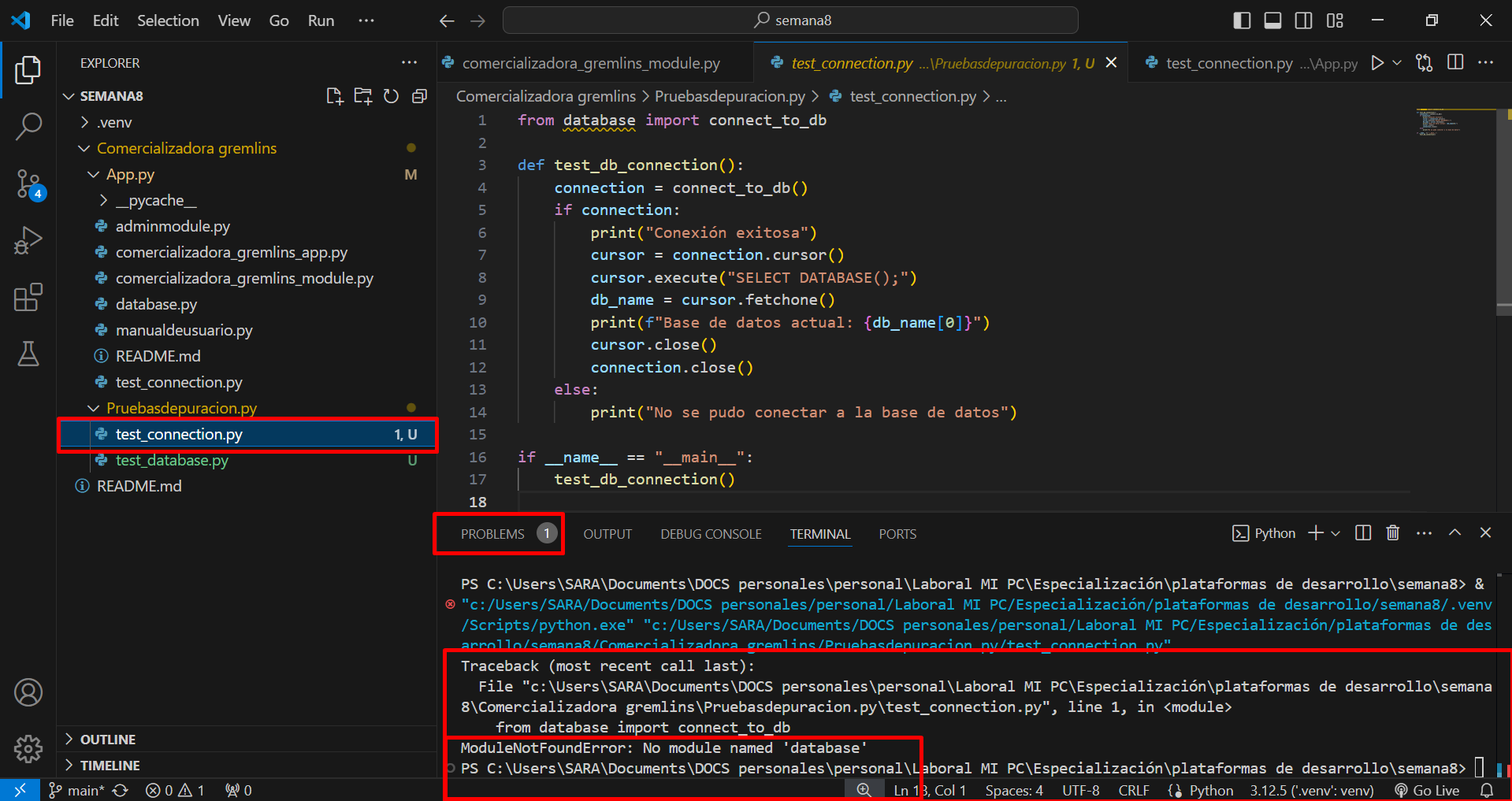
        print("No se pudo conectar a la base de datos")

if \_\_name\_\_ == "\_\_main\_\_":

    test\_db\_connection()

**Ejemplo 1. Se hizo una prueba fallida -**  El archivo test\_conecction.py no se ubicó correctamente y la conexión a la DB no era exitosa

**Imagen 19**. Resultado fallido de conexión a la DB

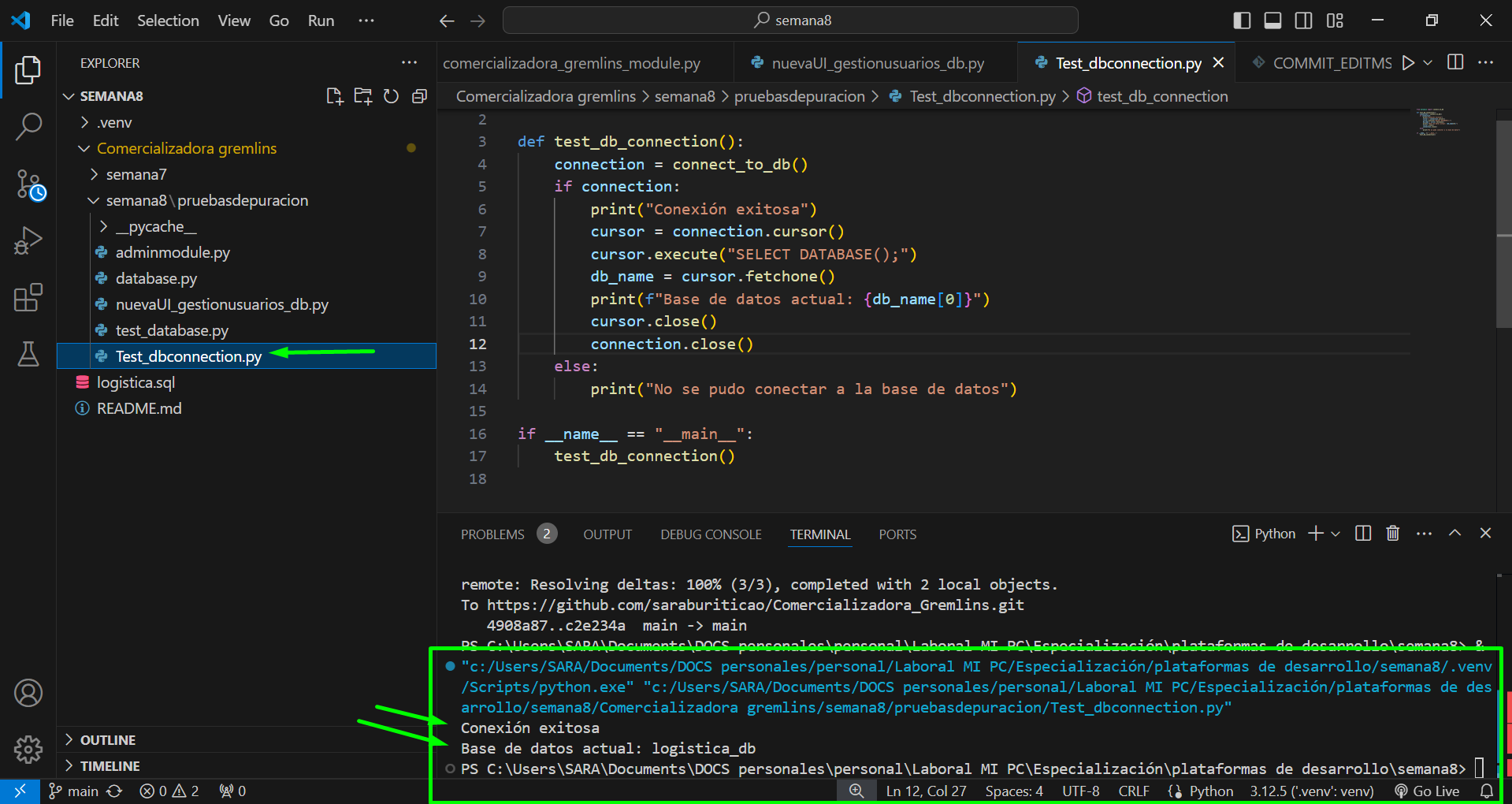


**Fuente:** propia

**Ejemplo 2.** Prueba exitosa de conexión a la DB

* **"Conexión exitosa"**: el script ha podido conectarse correctamente a la base de datos especificada. El programa ha verificado que los parámetros de conexión (como la dirección del servidor, las credenciales, etc.) son válidos y ha establecido la conexión sin errores.
* **"Base de datos actual: logistica\_db"**: Este mensaje indica que, después de establecer la conexión, el programa ha accedido correctamente a la base de datos llamada logistica\_db. Esto también confirma que la base de datos existe y que el usuario tiene acceso a ella.

**Imagen 20.** Resultado dePrueba exitosa de conexión a la DB

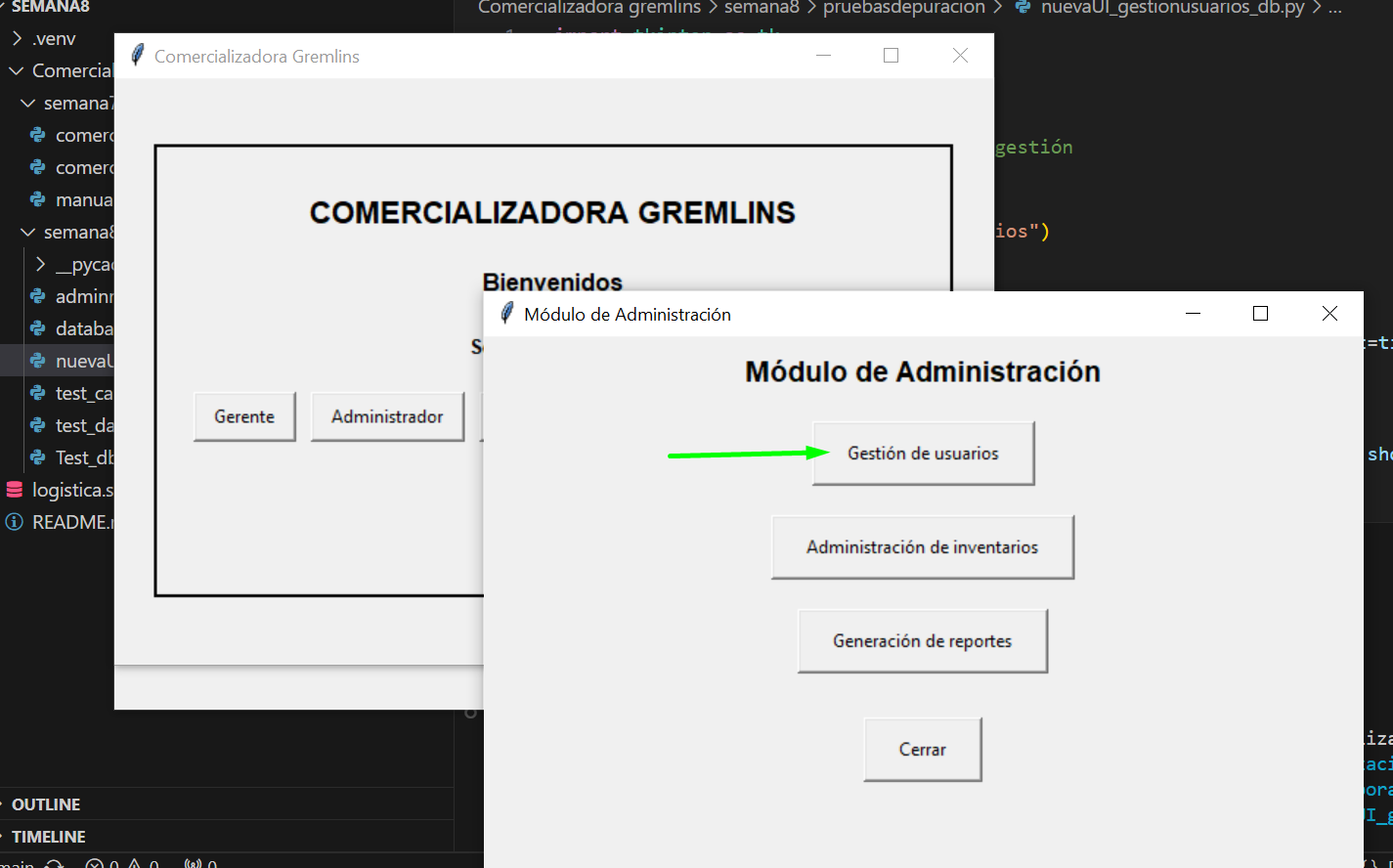


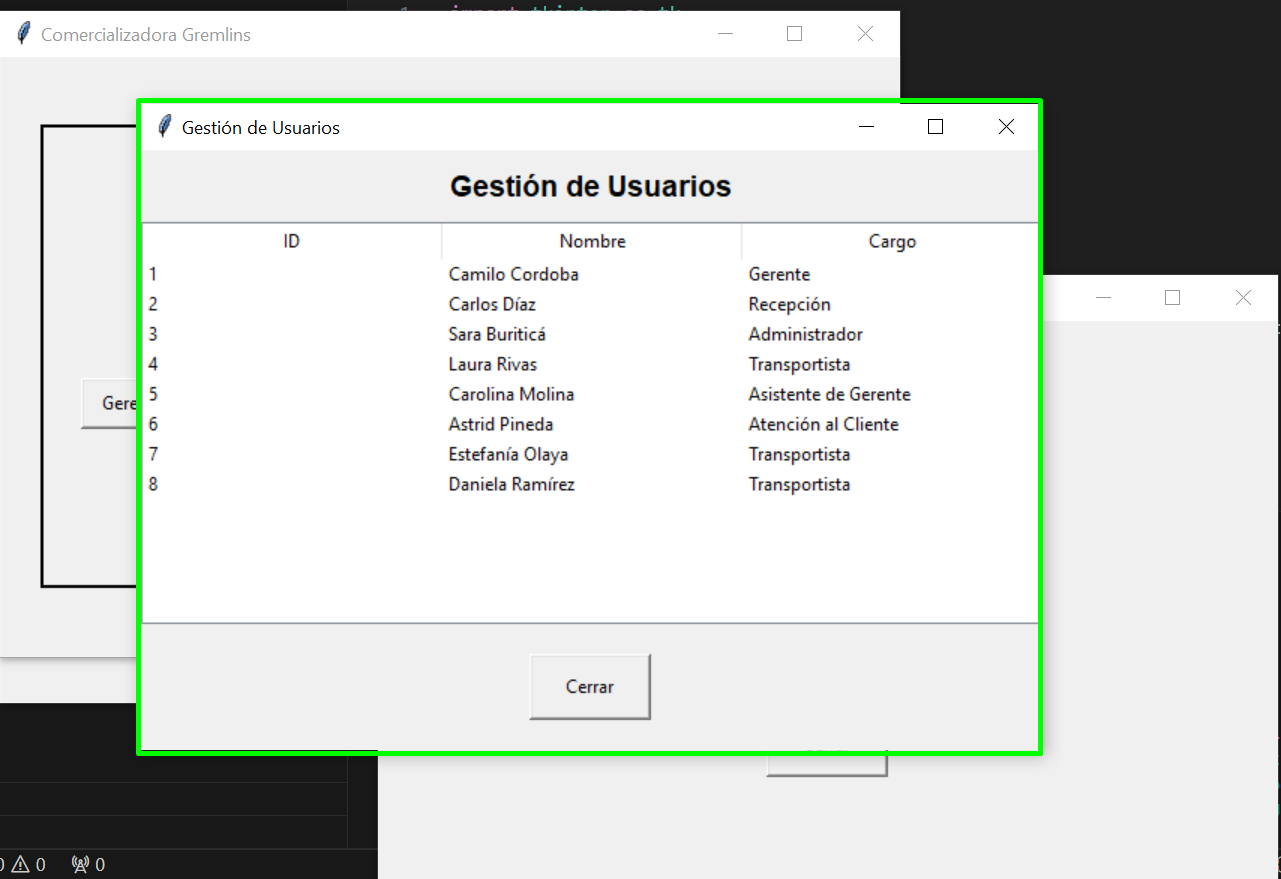
**Fuente:** Propia

**Pruebas funcionales de la interfaz de usuario (UI)**.

**Prueba Funcional de UI:**

**Ejemplo 1.** en este caso, la visualización de tabla de datos coincide con de la base de datos. Esto es visible en el módulo Admin > Gestionar usuario.

**Imagen 21.** Interfax gráfica de inicio



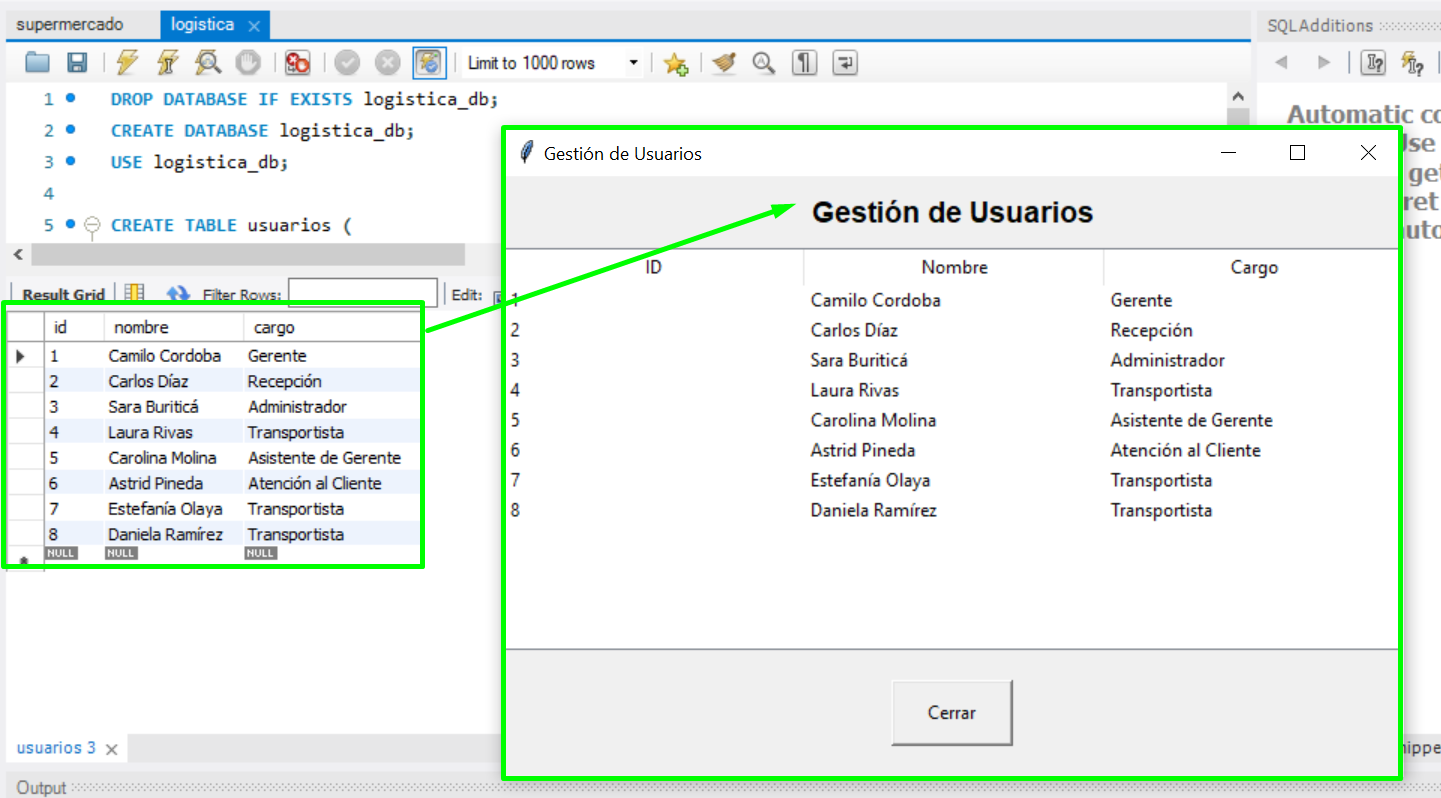
**Fuente:** propia

**Prueba de Integración:**

**Objetivo:** Verificar la funcionalidad de los nuevos componentes del sistema

**Ejemplo 1**. en este caso, la UI y la base de datos funcionan juntos de manera correcta.

**Imagen 22.** Compatibilidad entre la base de datos y los datos de la UI



**Fuente:** propia

**Prueba de Regresión:**

Se verifica quecambios en el código no hayan introducido errores en la funcionalidad existente de la UI.

**Ejemplo 1:** Al verificar la funcionalidad del aplicativo después de realizar cambios en el módulo de administración, que incluyeron la integración de la base de datos y la actualización del código de la interfaz de usuario (UI), se descubrió que la funcionalidad del sistema se había visto afectada, y los módulos ya no están funcionando correctamente

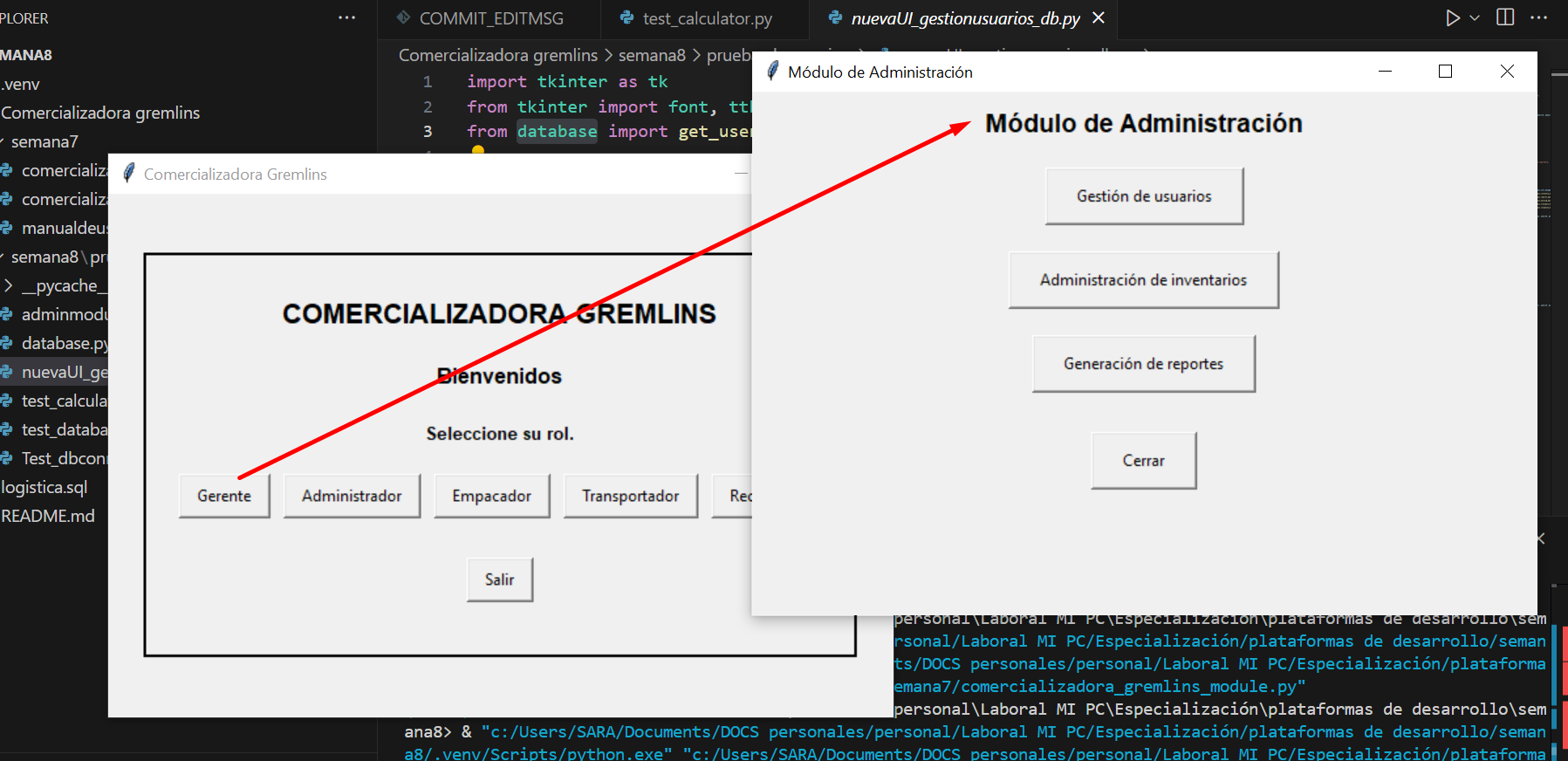
**Recordatorio :**

* Para la semana 7. la interfaz de usuario inicial se puede observar en el archivo comercializadora\_gremlins\_module.py.
* Para la semana 8. 7 la interfaz de usuario con errores se puede observar en el archivo nuevaUI\_gestionusuarios\_db.py
* la interfaz de usuario con la depuración aplicada se puede observar en el archivo depuraciondenuevaUI\_gestiondeusuarios\_db.py

para más claridad, ver imagen **Imagen 14.** Archivos en visual code.

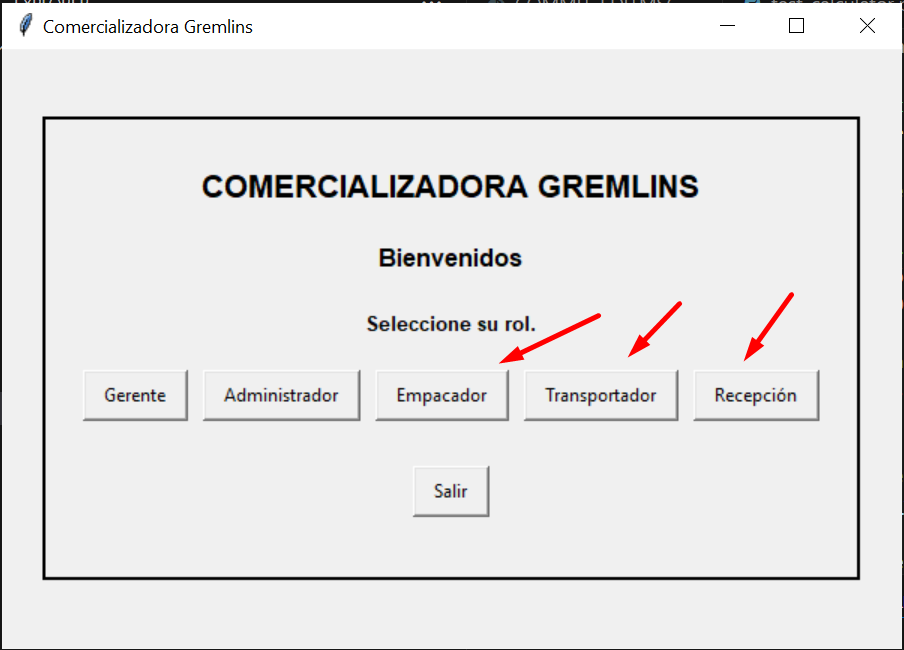
**Errores encontrados en la regresión**

**Imagen 23.** Error 1. - Al abrir el modulo Gerente se abre el módulo de administrador



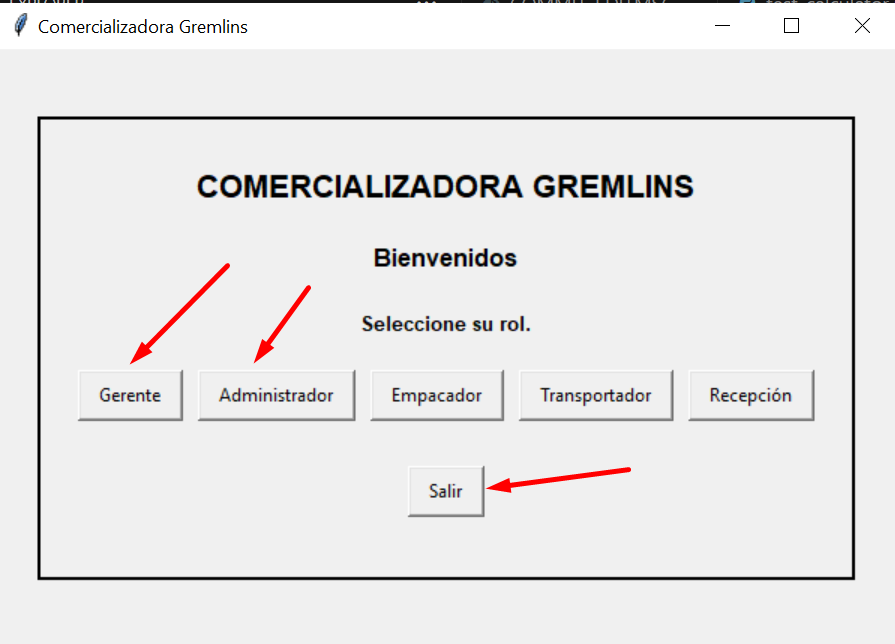
**Fuente:** propia

**Imagen 24.** Error 2 - Los módulos Empacador, transportador, y recepción ya no funcionan



**Fuente:** propia

**Imagen 25.** los botones para abrir los módulos de Gerente, Administrador y Salir funcionan



**Fuente:** propia

**Depuración de los errores encontrados.**

Se creó el archivo depuraciondenuevaUI\_gestiondeusuarios\_db.py para:

1. **Integración de Funcionalidades**: Se integraron las funcionalidades del primer código con las estructuras más completas del segundo código, para que todos los módulos y las funciones trabajen en conjunto sin problemas.
2. **Corrección de Llamadas a Funciones**: Se corrigieron las llamadas a funciones dentro de los botones y las ventanas de los módulos, asegurando que cada rol abra el módulo correcto.
3. **Eliminación de Duplicados y Simplificación**: Se eliminaron las partes redundantes y se simplificaron las llamadas a funciones para mejorar la legibilidad y evitar errores.
4. **Manejo de Errores y Validaciones**: Se añadió una validación de roles y una ventana emergente de mensaje (messagebox.showinfo) para manejar roles desconocidos o restringidos.
5. **Uso del Treeview**: El Treeview se mantiene para gestionar y visualizar datos en la ventana de gestión de usuarios.

import tkinter as tk

from tkinter import font, ttk, messagebox

from database import get\_users  # Asegúrate de que 'get\_users' esté correctamente implementado

# Funciones para abrir diferentes módulos basados en el rol

def open\_import\_management():

    create\_module\_window("Gestión de Importaciones - Gerente",

                         ["Clasificación y gestión de productos importados",

                          "Asignación y gestión de proveedores ocultos",

                          "Generación de informes"])

def open\_admin\_management():

    create\_module\_window("Módulo de Administración",

                         ["Gestión de usuarios",

                          "Administración de inventarios",

                          "Generación de reportes"],

                         open\_user\_management\_callback=open\_user\_management)

def open\_packing\_management():

    create\_module\_window("Módulo de Empaque",

                         ["Empaque de productos",

                          "Revisión de productos empacados",

                          "Actualización de inventario"])

def open\_transport\_management():

    create\_module\_window("Módulo de Transporte",

                         ["Gestión de rutas",

                          "Monitoreo de entregas",

                          "Registro de transportes"])

def open\_reception\_management():

    create\_module\_window("Módulo de Recepción",

                         ["Recepción de productos",

                          "Clasificación inicial",

                          "Registro de recibos"])

# Función genérica para crear una ventana de módulo

def create\_module\_window(title, options, open\_user\_management\_callback=None):

    module\_window = tk.Toplevel(root)

    module\_window.title(title)

    module\_window.geometry("600x400")

    title\_font = font.Font(family="Helvetica", size=14, weight="bold")

    title\_label = tk.Label(module\_window, text=title, font=title\_font)

    title\_label.pack(pady=10)

    for option in options:

        if option == "Gestión de usuarios" and open\_user\_management\_callback:

            button = tk.Button(module\_window, text=option, padx=20, pady=10, command=open\_user\_management\_callback)

        else:

            button = tk.Button(module\_window, text=option, padx=20, pady=10)

        button.pack(pady=10)

    close\_button = tk.Button(module\_window, text="Cerrar", command=module\_window.destroy, padx=20, pady=10)

    close\_button.pack(pady=20)

# Función para abrir la ventana de gestión de usuarios

def open\_user\_management():

    user\_window = tk.Toplevel(root)

    user\_window.title("Gestión de Usuarios")

    user\_window.geometry("600x400")

    title\_font = font.Font(family="Helvetica", size=14, weight="bold")

    title\_label = tk.Label(user\_window, text="Gestión de Usuarios", font=title\_font)

    title\_label.pack(pady=10)

    # Crear un árbol (Treeview) para mostrar los datos

    tree = ttk.Treeview(user\_window, columns=("ID", "Nombre", "Cargo"), show="headings")

    tree.heading("ID", text="ID")

    tree.heading("Nombre", text="Nombre")

    tree.heading("Cargo", text="Cargo")

    tree.pack(expand=True, fill="both")

    # Obtener los datos de la base de datos

    users = get\_users()

    for user in users:

        tree.insert("", tk.END, values=user)

    close\_button = tk.Button(user\_window, text="Cerrar", command=user\_window.destroy, padx=20, pady=10)

    close\_button.pack(pady=20)

# Selección del rol y acceso al módulo correspondiente

def on\_role\_selected(role):

    if role == "Gerente":

        open\_import\_management()

    elif role == "Administrador":

        open\_admin\_management()

    elif role == "Empacador":

        open\_packing\_management()

    elif role == "Transportador":

        open\_transport\_management()

    elif role == "Recepción":

        open\_reception\_management()

    else:

        messagebox.showinfo("Acceso Restringido", f"El rol '{role}' no tiene acceso a ningún módulo.")

# Función para salir de la aplicación

def on\_exit():

    root.destroy()

# Configuración de la ventana principal

root = tk.Tk()

root.title("Comercializadora Gremlins")

root.geometry("600x400")

# Configuración del título principal

frame = tk.Frame(root, padx=20, pady=20, relief="solid", bd=2)

frame.pack(expand=True)

titulo\_font = font.Font(family="Helvetica", size=16, weight="bold")

titulo = tk.Label(frame, text="COMERCIALIZADORA GREMLINS", font=titulo\_font)

titulo.pack(pady=10)

bienvenido\_font = font.Font(family="Helvetica", size=12, weight="bold")

bienvenido = tk.Label(frame, text="Bienvenidos", font=bienvenido\_font)

bienvenido.pack(pady=10)

# Configuración del mensaje de selección de rol

seleccion\_font = font.Font(family="Helvetica", size=10, weight="bold")

seleccion\_label = tk.Label(frame, text="Seleccione su rol.", font=seleccion\_font)

seleccion\_label.pack(pady=10)

# Configuración de los botones de rol

roles = ["Gerente", "Administrador", "Empacador", "Transportador", "Recepción"]

button\_frame = tk.Frame(frame)

button\_frame.pack(pady=10)

for rol in roles:

    button = tk.Button(button\_frame, text=rol, command=lambda r=rol: on\_role\_selected(r), padx=10, pady=5)

    button.pack(side=tk.LEFT, padx=5)

# Botón de salir

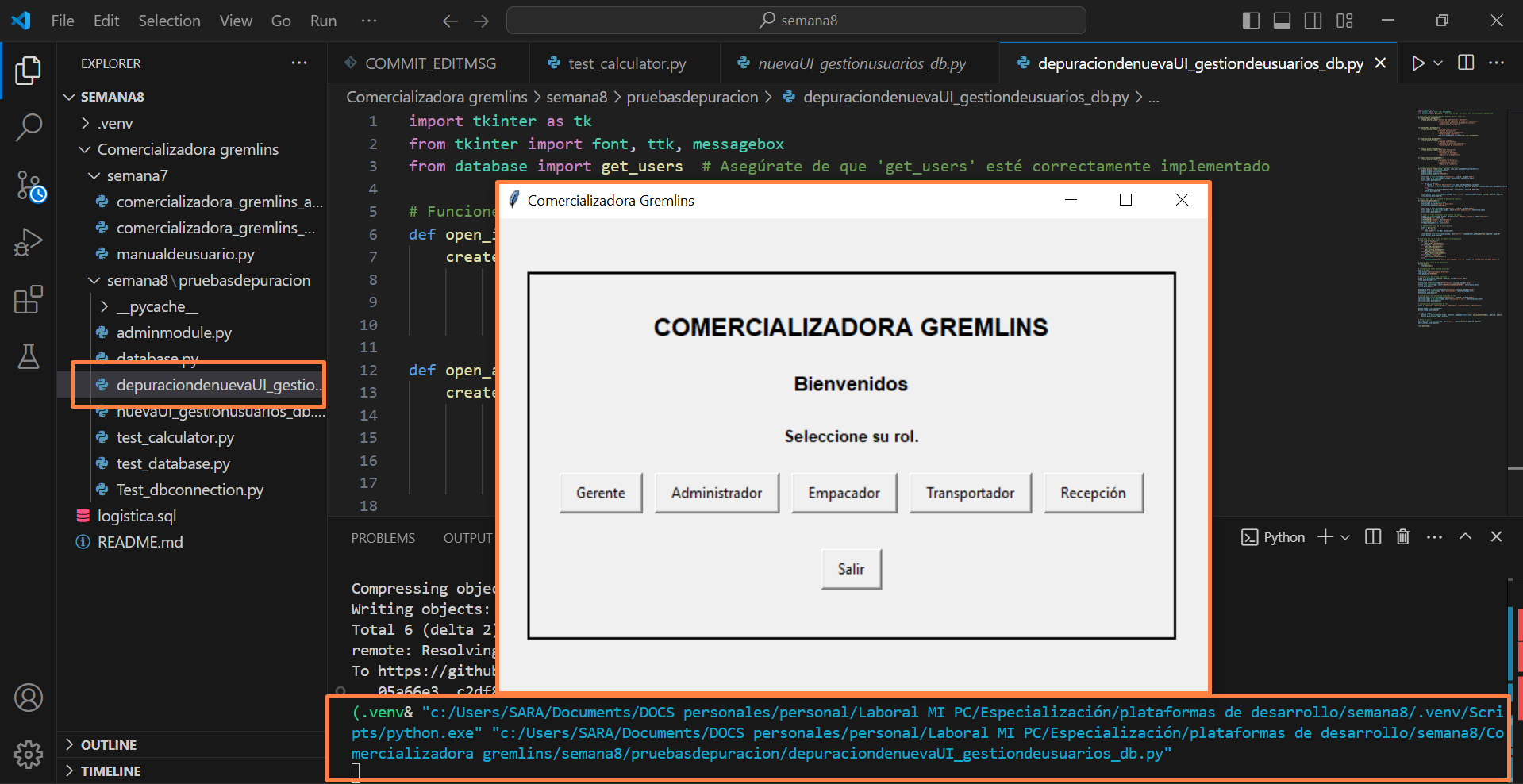
salir\_button = tk.Button(frame, text="Salir", command=on\_exit, padx=10, pady=5)

salir\_button.pack(pady=20)

root.mainloop()

Con estas mejoras, el código funciona correctamente, permitiendo abrir y gestionar los distintos módulos según el rol seleccionado.

**Imagen 26.** Interfaz de usuario depurada y funcional



**Fuente:** propia

**Conclusiones**

1. La implementación de la aplicación utilizando Python y la biblioteca tkinter permitirá a la Comercializadora Gremlins optimizar su proceso logístico de manera significativa. Al abordar las ineficiencias actuales, esta solución fortalecerá la posición de la empresa en el mercado, proporcionando un servicio más eficiente y adaptado a las expectativas modernas de los clientes. Con la adopción de esta tecnología, se anticipa una mejora notable en la gestión operativa, contribuyendo al crecimiento y la sostenibilidad de Gremlins en un entorno de negocios competitivo y en constante cambio.
2. El desarrollo de esta aplicación para la Comercializadora Gremlins no solo optimiza los procesos logísticos actuales, sino que también demuestra el poder de tkinter como una solución práctica y accesible para crear interfaces gráficas en Python. La aplicación resultante no solo mejora la eficiencia operativa, sino que también posiciona a Gremlins de manera más competitiva en el mercado, ayudando a garantizar su éxito y sostenibilidad a largo plazo en un entorno empresarial dinámico.
3. El proceso de pruebas y depuración en el desarrollo de esta aplicación ha sido crucial para garantizar su funcionalidad y eficiencia. A través de un enfoque metódico en la identificación y resolución de errores, no solo se asegura que la aplicación cumpla con los requisitos establecidos, sino que también minimiza el riesgo de fallos en el entorno operativo real, protegiendo así la inversión de la Comercializadora Gremlins y asegurando una mayor satisfacción del usuario final.
4. La implementación de técnicas de pruebas y depuración ha demostrado ser fundamental en la entrega de un software de calidad, eficiente y adaptado a las necesidades de la empresa. Al detectar y corregir errores de manera temprana, se reduce significativamente el tiempo de desarrollo y se mejora la estabilidad del sistema, lo que contribuye a un ciclo de vida del software más saludable y rentable.

**Referencias**

Maida, E. G., & Pacienzia, J. (2015). Metodologías de desarrollo de software.

<https://repositorio.uca.edu.ar/bitstream/123456789/522/1/metodologias-desarrollo-software.pdf>

Cardila, A. (n.d.). Principios de pruebas: Enfoques estratégico y operativo [PDF]. Universidad del Cauca. <http://artemisa.unicauca.edu.co/~cardila/CS__Cap_3.1__Principios_de_pruebas__Enfoques_estrategico_y_operativo.pdf>