

Criterio C: Desarrollo

Metodología

Herramientas Utilizadas

1. **Base de Datos Relacional:** En MySQL se creó una base de datos de tipo relacional en la que se almacenaron los datos pertinentes para el funcionamiento del programa: la información de los clientes, de los empleados, de los servicios que ofrece la barbería, así como las citas programadas.
2. **Lenguaje de Programación Python:** Se utilizó el lenguaje de programación Python para la creación del código fuente del programa, que permite recuperar la información plasmada en la base de datos, así como crear las funciones necesarias para hacer posible la gestión de los datos anteriores.
 - 2.1. **Programación Orientada a Objetos:** Se implementó la POO al crear una *clase Producto* que inicializa el sistema en su totalidad a partir de la creación de la ventana principal.
3. **Biblioteca Gráfica Tkinter:** Dentro de Python, se accedió a la biblioteca Tkinter con el objetivo de crear la interfaz gráfica a la que el cliente accede para hacer uso del programa, de modo que incluya las funcionalidades especificadas los criterios anteriores, y ser de fácil acceso para los usuarios.

Base de Datos

clientes		
Atributo	Variable	Descripción
ID_cliente	int PK	Llave primaria que funciona como identificador único del cliente.
nombre_cliente	varchar(45)	Indica el nombre y apellido del cliente.
contacto	double	Almacena el número de teléfono del cliente.

Figura 1. Entidad *clientes*.

servicios		
Atributo	Variable	Descripción
ID_servicio	int PK	Llave primaria que funciona como identificador único del servicio.
nombre_servicio	varchar(45)	Indica el nombre del servicio.
precio	varchar(45)	Indica el precio de cada servicio; se utiliza varchar para permitir el uso del \$.
disponible	varchar(45)	Señala la disponibilidad del servicio, que puede alterarse en caso de que en algún momento no lo esté (Si/No).

Figura 2. Entidad *servicios*.

citas		
Atributo	Variable	Descripción
ID_cita	int PK	Llave primaria que funciona como identificador único de la cita.
ID_cliente	int FK	Llave foránea que atribuye una cita con un cliente en específico.
ID_servicio	int FK	Llave foránea que relaciona la cita con el servicio que el cliente solicita.
ID_empleado	int FK	Llave foránea que asigna un empleado disponible en el horario en que se solicita la cita.
horario	time	Indica el horario de 9:00 a 20:00 en que se programa la cita, de modo que se relacione con los empleados disponibles según el día.
fecha	date	Registra la fecha exacta, registrada en formato AAAA/MM/DD para agendar la cita.
estatus	varchar(45)	Distingue entre <i>Pendiente</i> , <i>Completada</i> y <i>Cancelada</i> , para indicar el estatus de la cita según la fase en la que se encuentre.

Figura 3. Entidad *citas*.

empleados		
Atributo	Variable	Descripción
ID_empleado	int PK	Llave primaria que funciona como identificador único del empleado.
usuario_empleado	varchar(45)	Almacena el usuario con el que el empleado tiene acceso al programa de citas.
contraseña_empleado	varchar(45)	Almacena la contraseña correspondiente al usuario para el acceso al programa.
perfil	varchar(45)	Se distingue entre <i>Empleado</i> y <i>Administrador</i> , de modo que cada uno de ellos tendrá acceso a diferentes funciones.
nombre_empleado	varchar(45)	Indica el nombre completo del empleado.
día_descanso	varchar(45)	Indica el día de descanso del empleado, ya que cada uno descansa en un día de la semana distinto.

Figura 4. Entidad *empleado*.

Desarrollo de Criterios de Logro

1. Introducción de nuevas citas por parte de los administradores, que incluyan el nombre del cliente, el servicio deseado, y el horario de la cita: En la pestaña de *Nuevas Citas*, el usuario selecciona los elementos de la base de datos (cliente/servicio), e introduce el horario, los cuáles al presionar el botón, se guardan en la base de datos, y también valida el formato de la fecha con una función.

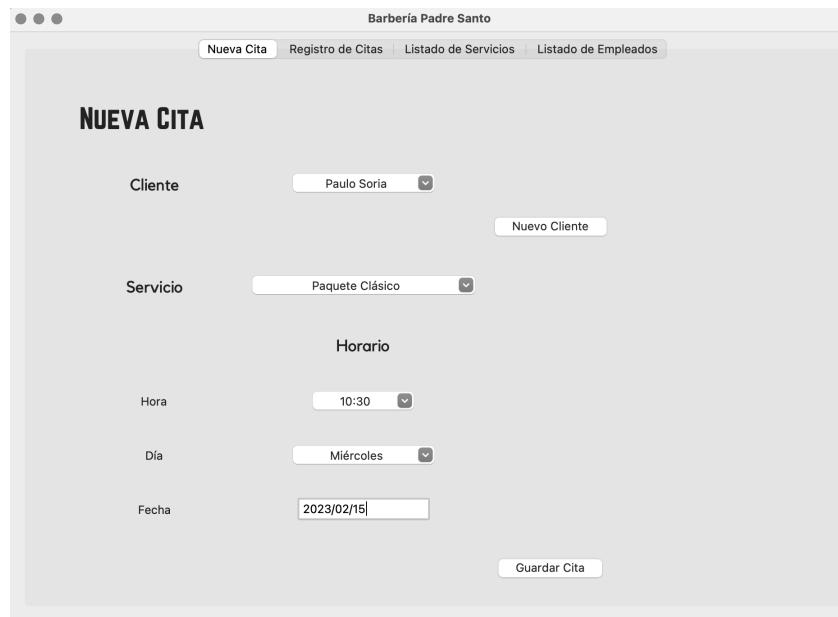


Figura 5. Pestaña *Nueva Cita*.

```
#el formato de la fecha
def validar_fecha():
    while True:
        try:
            datetime.strptime(entry_fecha.get(), '%Y/%m/%d')
            #fecha válida
            comprobar_empalmes()
            break
        except ValueError:
            #fecha inválida
            ttk.Label(frame_rcita, text="Error", font=("Norwester", 35, 'bold')).grid(column=0, row=2, columnspan=2, ipadx=20, ipady=20, sticky= 'E')
            ttk.Label(frame_rcita, text = "Ingrese la fecha en \n el formato:", font= ("Kollektiv", 20)).grid(column=1, row=3, columnspan=2, ipadx=15, ipady=20, sticky= 'NS')
            ttk.Label(frame_rcita, text = "(AAAA/MM/DD)", font=(14)).grid(column=2, columnspan=2, row=4, ipadx=15, ipady=20, sticky= 'NS')
            break
    continue
```

Figura 6. Función para validar fecha.

2. Acceso al programa, de modo que tanto administradores como empleados puedan hacer

uso del mismo: Se creó una función para recuperar los usuarios y contraseñas de los administradores, validarlos con los que introduzca el cliente, y no dar acceso a las siguientes pantallas en caso de no coincidir.



Figura 7. Ventana *Inicio de Sesión*.

```
#BOTÓN INICIO SESIÓN
def boton_inicio():
    acceso = False
    print ("\\n\\n INICIO DE SESIÓN")
    usuario = entry_usuario.get() #entry de tkinter
    contraseña = entry_contra.get()

    sqlQuery1 = "select * from empleados where usuario_empleado = %s and contraseña_empleado = %s"
    cursor_empleados.execute(sqlQuery1, [usuario,contraseña])
    inicio_sesión = cursor_empleados.fetchall()

    if len (inicio_sesión):
        acceso = True
        raise_frame(frame_menu)
        for a in inicio_sesión:
            print ("\n\n\t >> Bienvenidx", a[3], a[4], "``")
            id_empleado = (a[0])
    else:
        #texto en tkinter
        print("\tACCESO DENEGADO")

bt1=ttk.Button(frame_i, text="Iniciar Sesión", command=boton_inicio).grid(column=3, row=8,pady=20)
```

Figura 8. Función inicio de sesión

3. Listado de servicios que ofrece la barbería para poder ser consultados y registrados al momento de hacer una cita, así como añadir novedades y modificar su disponibilidad en caso de requerirse: Para mostrar los atributos de *servicios*, el *treeview* se llenó con un *for* que recorre *servicios* e inserta los valores en los cuadros conforme el índice *i* de la posición incrementa. Al botón *Editar Servicios*, mediante *raise*, se eleva el *frame* que permite añadir un nuevo servicio y modificar la disponibilidad.



Figura 9. Pestaña *Listado de Servicios*, y su configuración *Editar Servicios*.

```
def servicios_todos():
    #recuperar servicios base de datos
    sqlQuery2 = "select * from servicios"
    cursor_servicios.execute(sqlQuery2)
    tabla_servicios = cursor_servicios.fetchall()
    return tabla_servicios

tabla_servicios = servicios_todos()
for n in tabla_servicios:
    tr2.insert("", tk.END, values=(n[0],n[1],n[2],n[3]))

bt3=ttk.Button(frame_serv, text="Editar Servicios", command=gest_servicio).grid(column=0, row=3,pady=40, padx=20, columnspan=2, stick='e')
```

Figura 10. Conexión a *servicios* para llenar *Listado Servicios*.

- 4. Consulta del registro de citas con la información, con posibilidad de modificar las citas en caso de que un cliente haga la cancelación:** Se obtiene la información de *citas*, que relaciona mediante *foreign keys* a un cliente, servicio, y empleado una sola cita. Se utilizó *inner join* para juntar las entidades mediante su ID para mostrar los nombres. Incluye un botón que dirige mediante un comando la función para modificar el estado de la cita (Criterio 7).

Folio	Cliente	Servicio	Fecha	Horario	Empleado	Estado
1	Mauricio Ramírez	Corte de Cabello	2023-01-30	11:00:00	Alexis	Completado
2	Bruno Verduzco	Paquete Clásico	2023-01-30	12:15:00	Fernando	Completado
3	Paulo Soria	Corte de Cabello Niño	2023-01-30	17:00:00	Erick	Cancelado
4	Alejandro Lara	Afeitado de Cabeza	2023-02-01	9:00:00	Fernando	Completado
6	Miguel Bravo	Tinte Cabello	2023-02-01	13:45:00	Erick	Cancelado
7	Esteban Carbaljal	Parches de Colágeno Anti-Ojeras	2023-02-02	16:30:00	Daniel	Completado
8	Oscar Parra	Paquete Silver	2023-02-03	12:45:00	Fernando	Completado
9	Javier Mora	Corte Express	2023-02-03	14:30:00	Erick	Completado
10	Carlos Jiménez	Corte de Cabello	2023-02-03	16:15:00	Daniel	Completado
11	Rodrigo Gómez	Facial Limpieza Profunda	2023-02-04	9:30:00	Alexis	Completado
12	Juan Mendoza	Recorte y Alineado de Bigote	2023-02-04	13:00:00	Gabriel	Cancelado
13	Alberto Rivas	Paquete Padre Santo VIP	2023-02-04	14:30:00	Daniel	Completado
14	Nicolás Morales	Mascarilla Negra de Yogurt	2023-02-04	18:15:00	Erick	Completado
25	Alberto Rivas	Corte Express	2023-02-04	9:00:00	Por Definir	Pendiente

Figura 11. Pestaña con *Registro de Citas*.

```
#INNER JOIN REGISTRO DE CITAS
def inner_join():
    sqlCommand = "select * from citas C INNER JOIN clientes L ON C.ID_cliente = L.ID_cliente [INNER JOIN servicios S ON C.ID_servicio = S.ID_servicio INNER JOIN empleados E ON C.ID_empleado = E.ID_empleado"
    cursor_citas.execute(sqlCommand)
    registro_citas = cursor_citas.fetchall()
    return registro_citas

#recuperar citas base de datos
registro_citas = inner_join()
for m in registro_citas:
    tr1.insert("",tk.END, values=(m[0],m[9],m[12],m[5],m[7],m[4],m[19],m[6]))
bt5=ttk.Button(frame_citas, text="Actualizar Estado", command=act_estado).grid(column=0, row=3,pady=55, padx=20, columnspan=2, stick='e')
```

Figura 12. *Inner Join* para juntar las entidades y llenar *Registro de Citas*.

5. Visualización de los empleados disponibles en la barbería, de sus horarios de disponibilidad para poder asignar las citas que les corresponde a cada uno: Se coloca el día descanso para evitar que se agenden citas cuando no se encuentre. Se replicó el proceso del Criterio 3, donde recuperaron la información de *empleados* de la base de datos, excluyendo a los administradores mediante un *where*.



Figura 13. Pestaña con *Listado de Empleados*.

6. Introducción de los empleados en cada horario, así como distribución de las citas agendadas:

Se seleccionó el *ID_empleado* a partir del nombre, y mediante un *update* se modifica en la entidad *citas*. Asimismo, es posible introducir a la base de datos nuevos empleados con *insert*. En caso de que el empleado esté ocupado o ausente, se mostrará el error. Los cambios se ven reflejados en el *Registro de Citas* y *Listado de Empleados* al vaciar su contenido previo e insertarlo nuevamente.

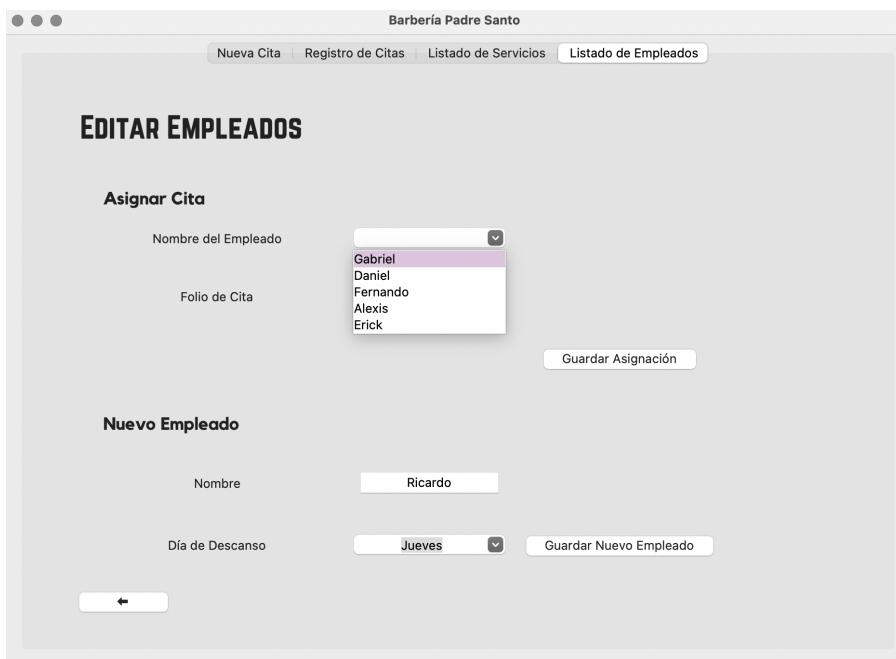


Figura 14. Ventana para *Editar Empleados*.

```
#update idempleado con folio de cita
def asignar_empleado():
    sqlQueryU = (f"update citas_barberia.citas SET ID_empleado = '{var_idemploy}' where ID_cita = '{entry_folioemploy.get()}'")
    cursor_empleados.execute(sqlQueryU)
    mydb.commit()
    tr1.delete(*tr1.get_children())
    #volver a llenar el treeview"
    registro_citas = inner_join()
    for m in registro_citas:
        tr1.insert('',tk.END, values=(m[0],m[9],m[12],m[5],m[4],m[19],m[6]))
    #raise frame
    raise_frame(frame_employ)
```

Figura 15. *Select* y *Update* para asignar citas a un empleado.



Figura 16. Posibles *Errores* al asignar empleados

```
#no exista una cita con un mismo empleado, fecha y hora
def comprobar_empalmes():
    sqlQueryE = "select * from citas where horario = '" + str(var_hcita) + "' and ID_empleado = " + str(var_idemploy) + " and fecha = '" + str(var_fcita) + "'"
    cursor_citas.execute(sqlQueryE)
    array_empalme(cursor_citas.fetchall())
    if not array_empalme:
        #no hay coincidencias
        asignar_empleado()
    else:
        #empalme
        frame_errorres()
        ttk.Label(frame_eemploy, text="Error", font=("Norwester", 35, 'bold')).grid(column=0, row=2, ipadx=20, ipady=20, columnspan=2, sticky='E')
        ttk.Label(frame_eemploy, text = "El empleado ya tiene una cita en esa\nfecha en ese horario.", font=("Kollektif", 20)).grid(column=1, columnspan=2, row=3, ipadx=15, ipady=20, sticky='NS')
        ttk.Label(frame_eemploy, text = "Intente algún otro empleado.", font=("Kollektif", 25)).grid(column=2, row=4, columnspan=2, ipadx=15, ipady=20, sticky='NS')
        raise_frame(frame_eemploy)

def dia_descanso():
    if var_dcita == var_demploy:
        frame_errorres()
        ttk.Label(frame_eemploy, text="Error", font=("Norwester", 35, 'bold')).grid(column=0, row=2, ipadx=20, ipady=20, columnspan=2, sticky='E')
        ttk.Label(frame_eemploy, text = "El empleado no trabaja ese día", font=("Kollektif", 20)).grid(column=1, columnspan=2, row=3, ipadx=15, ipady=20, sticky='NS')
        ttk.Label(frame_eemploy, text = "Intente algún otro empleado.", font=("Kollektif", 25)).grid(column=2, row=4, columnspan=2, ipadx=15, ipady=20, sticky='NS')
        raise_frame(frame_eemploy)
        print("no labora")
    else:
        print("si labora el employ")
        comprobar_empalmes()
```

Figura 17. *Select* en *citas* para detectar coincidencias y condicional *If* para evaluar día de la cita y descanso del empleado.

7. Indicador de las citas una vez atendidas por parte de los empleados donde administradores y empleados identifiquen el estado de la cita como pendiente, completada, o cancelada: Es posible ingresar el folio de la cita (*ID_cita*), y seleccionar el nuevo estado, de modo que se obtienen dichos valores con la función *get()*, y utilizando el *ID* como referencia, se utiliza la cláusula *update* para modificar la columna *estatus*.



Figura 18. Ventana para *Actualizar Estado* de citas.

8. Registro de los datos de los clientes nuevos para agilizar el registro de nuevas citas para clientes anteriores: El botón *Nueva Cliente* sobrepone el frame que permite ingresar los nuevos datos. Mediante *insert*, se obtienen sus valores y se colocan en las columnas de la entidad *clientes*. Una vez presionado el botón, el usuario verá el cliente recién registrado en la *combobox* de *Nueva Cita*.



Figura 19. Ventana para agregar *Nueva Cliente*.

9. Indicador de empalme en caso de que se intente agendar una cita en un horario saturado por citas registradas con anterioridad: Se validan los datos antes introducirlos en la base de datos para no registrar citas cuando esté cerrado, y corroborar que no exista una cita con un mismo servicio, fecha y hora. En caso de no cumplirse, tras presionar *Guardar Cita*, se indica el error correspondiente para que pueda corregirse.

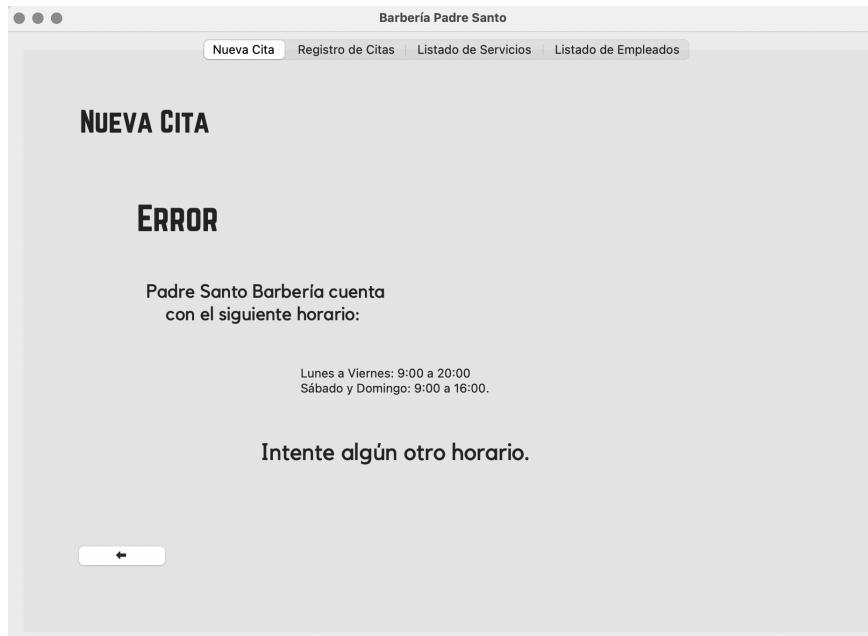


Figura 20. Ventana *Error* por cita fuera del horario.

```
#citas en sábado y domingo antes de las 15:00
def horario_fines():
    #if dia == sábado or domingo and hora == 15,16,17,18,19
    if var_dia.get() == "Sábado" or var_dia.get() == "Domingo":
        if var_hora.get() == "15:15" or var_hora.get() == "15:30" or var_hora.get() == "15:45" or var_hora.get() == "16:00" or var_hora.get() == "16:15" or var_hora.get() == "16:30"
            #horario no válido, fin de semana
            ttk.Label(frame_rcita, text="Error", font="Times New Roman", 35,'bold')).grid(column=0, row=2,columnspan=2,ipadx=20,ipady=20,sticky= 'E')
            ttk.Label(frame_rcita, text = "Padre Santo Barbería cuenta \n con el siguiente horario:", font="Sans Serif",20)).grid(column=1,row=3,columnspan=2,ipadx=15,ipady=20,
            ttk.Label(frame_rcita, text = "Lunes a Viernes: 9:00 a 20:00\nSábado y Domingo: 9:00 a 16:00.", font=(14)).grid(column=2, row=4,columnspan=2,ipadx=15,ipady=20,sticky= 'E')
            ttk.Label(frame_rcita, text = "Intenta algún otro horario.", font="Sans Serif", 25)).grid(column=2, row=5,columnspan=2,ipadx=15,ipady=20,sticky= 'NS')

    else:
        #sábado con horario válido
        validar_fecha()
    else:
        #entre semana, horario válido
        validar_fecha()
```

Figura 21. Condicional *If* para evaluar *Día* y *Hora* ingresados por el usuario.



Figura 22. Ventana *Error* por empalme de citas.

```
"""VALIDACIONES"""
#no exista una cita con un mismo servicio, fecha y hora
def comprobar_empalmes():
    sqlQueryE = "select * from citas where horario = '" + (var_hora.get()) + "' and ID_servicio = " + str(var_idserv) + " and fecha = '" + str(entry_fecha.get()) + "'"
    cursor_citas.execute(sqlQueryE)
    array_empalme=cursor_citas.fetchall()
    if not array_empalme:
        #no hay coincidencias
        insertar_cita()
    else:
        #empalme
        ttk.Label(frame_rcita, text="Error", font=("Times New Roman", 35, 'bold')).grid(column=0, row=2, ipadx=20, ipady=20, columnspan=2, sticky='E')
        ttk.Label(frame_rcita, text = "Empalme por saturación de\\horarios en citas registradas", font=("Sans Serif",20)).grid(column=1, columnspan=2, row=3, ipadx=15, ipady=20, sticky='NS')
        ttk.Label(frame_rcita, text = "Intente algún otro horario.", font=("Sans Serif", 25)).grid(column=2, row=4, columnspan=2, ipadx=15, ipady=20, sticky='NS')
```

Figura 23. *Select* de citas con los parámetros ingresados para descartar empalmes.

(987 Palabras)