

**Documentación del Sistema de Reservaciones para el  
Hotel Le Villa**



La empresa “Hotel Le Villa” desea implementar un sistema informático que permita gestionar de manera eficiente las reservaciones, habitaciones, clientes, servicios adicionales y pagos del establecimiento. El sistema debe representar a las distintas entidades que forman parte de las operaciones del hotel, considerando que cada una cumple un rol diferente dentro del proceso de administración y atención al cliente. Además, se requiere llevar un registro de fechas importantes, como la fecha de ingreso y salida de cada cliente, la fecha de realización de una reserva o la fecha de pago. El equipo de desarrollo ha establecido que el sistema debe cumplir con las siguientes condiciones:

- Todas las habitaciones comparten atributos básicos, como el número de habitación, tipo (sencilla, doble, suite) y estado (disponible, ocupada, en mantenimiento).
- Los clientes poseen información particular, como su nombre, número de identificación, número de contacto y el historial de reservaciones realizadas.
- Cada reserva debe incluir los datos del cliente asociado, la habitación asignada, las fechas de entrada y salida, así como los servicios adicionales solicitados.
- Los servicios adicionales (como restaurante, spa o transporte) deben poder registrarse y asociarse a una reserva específica.
- Los pagos deben reflejar el monto total, método de pago y fecha de transacción.
- Todas las entidades deben poder mostrar su información en un formato legible y estructurado.

## **Descripción general**

El presente proyecto tiene como propósito desarrollar un sistema de reservaciones para un hotel, que permita automatizar y optimizar la gestión de habitaciones, clientes, servicios y pagos.

### **Objetivo general:**

Automatizar la gestión de reservaciones del hotel para mejorar la eficiencia operativa y la satisfacción del cliente.

### **Objetivos específicos:**

- Optimizar la asignación y disponibilidad de habitaciones en tiempo real.
- Registrar y administrar los datos de los clientes y sus reservaciones.
- Controlar los servicios adicionales y pagos asociados a cada estancia.
- Reducir errores manuales y aumentar la trazabilidad de las operaciones.
- Proveer reportes claros sobre ocupación, ingresos y actividad del hotel.

### **Actores del sistema**

## Actores principales

- **Cliente (Customer):** El actor más importante. Es la persona que busca y realiza reservas de habitaciones, reserva servicios adicionales y efectúa pagos. Sus interacciones directas giran en torno a su propia cuenta, sus reservas y sus pagos.
- **Empleado del Hotel (Staff):** Este es un actor general que se subdivide en roles más específicos que interactúan directamente con el sistema para gestionarlo:
  - **Repcionista:** Actor principal dentro del personal, encargado de crear, modificar y cancelar reservas de habitaciones, gestionar el *check-in/check-out* y manejar los detalles de los clientes. (Visto en `receptionist.py` y `employee.py`).
  - **Botones (Bellboy):** Otro tipo de empleado que interactúa con el sistema para registrar o notificar la finalización de servicios (implícito en `execute.py` y `populate_db.py`)

## Actores secundarios

- **Administrador del Sistema / Gerente (Manager):** Un rol de empleado de alto nivel (visto en `employee_dao.py`) que es responsable de la administración, configuración inicial y el mantenimiento de la integridad de los datos (p.ej., la creación de empleados, la gestión de la tabla de servicios, la población inicial de datos a través de `populate_db.py`).
- **Sistema de Gestión de Base de Datos (MySQL):** Un componente de *backend* vital (visto en `db_connection.py` y todos los archivos `*_dao.py`). Actúa como actor de soporte al ser el responsable de almacenar, recuperar y asegurar la persistencia de todas las reservas, clientes, habitaciones y pagos del sistema.

## Actores externos

- **Pasarela de Pago / Sistema Bancario:** La entidad financiera o servicio tecnológico (API) que se encarga de procesar y validar la transferencia real de fondos para el objeto Payment. El sistema de gestión solo registra el método y el monto, pero la pasarela externa confirma la transacción.
- **Servicio de Entorno/Configuración (OS/Environment):** El entorno operativo que proporciona las variables de configuración sensibles y críticas (como las credenciales de la base de datos HOST, USER, PASSWORD) a través del archivo `mysql_env.py` para que el sistema pueda arrancar y conectarse a su persistencia.
- **Sistema de Tiempo y Fecha:** El componente que proporciona la hora y fecha actual (utilizado por `datetime` en `payment.py` y para las fechas de *check-in/check-out*). Es fundamental para registrar con precisión los eventos y pagos.

## Principios de POO aplicados en el sistema

A continuación, se detalla la aplicación de los cuatro pilares de la POO en el código de su sistema:

### 1. Encapsulamiento (Encapsulation)

El encapsulamiento se evidencia en la forma en que las clases controlan el acceso a sus datos internos, manteniendo la **integridad del estado** de cada objeto.

- **Evidencia en el Código:** En casi todas las clases de entidad (customer.py, employee.py, payment.py, reservation.py), los atributos clave están declarados como **privados** mediante el convenio de Python del doble *underscore* (\_), por ejemplo: self.\_id, self.\_name, self.\_password.
- **Mecanismo de Control:** Se proporcionan métodos públicos de acceso y modificación, conocidos como **Getters** (getId(), getName()) y **Setters** (setId(), setName()).
- **Justificación Académica:** Esta práctica asegura que la lógica de negocio separe el acceso a los datos de su representación interna. Al forzar el uso de **setters**, el desarrollador tiene puntos específicos donde puede incluir **validaciones o reglas de negocio** (p.ej., verificar que un nuevo email sea válido) antes de que el estado del objeto sea alterado, previniendo la corrupción de los datos.

### 2. Herencia (Inheritance)

La herencia se utiliza para establecer una **relación taxonómica** de "es un" (*is-a*) entre las clases, promoviendo la reutilización de código y la especialización de roles.

- **Evidencia en el Código:** El archivo receptionist.py (y de forma análoga, la clase Bellboy que se usa en execute.py y populate\_db.py) extiende la clase base Employee:

Python

```
class Receptionist(Employee):  
    # ... métodos especializados
```

- **Impacto en el Diseño:** Al heredar de Employee, las clases Receptionist y Bellboy obtienen automáticamente todos los atributos básicos (nombre, email, CURP, etc.) y comportamientos generales del empleado. Esto evita la duplicación de código en la definición de estas subclases.
- **Justificación:** La herencia permite modelar la jerarquía organizacional del hotel. La clase Employee sirve como el **tipo base**, y las subclases especializan el comportamiento (p.ej., un Receptionist tiene métodos específicos para

interactuar con la reserva, como `createReservation` y `modifyReservation`, aunque actualmente llaman a `super()`, la estructura está lista para una futura especialización).

### 3. Abstracción (Abstraction)

La abstracción se manifiesta al ocultar la complejidad operacional, permitiendo que las interacciones se realicen a través de interfaces bien definidas y de alto nivel.

- **Evidencia en el Código:** El mejor ejemplo de abstracción es la **Capa de Acceso a Datos (DAO)**, con clases como `CustomerDAO`, `ReservationDAO`, `PaymentDAO`, etc.
  - Una parte del sistema (p.ej., la vista de la interfaz gráfica o un controlador) solo necesita llamar a: `customer_dao.create(new_customer)`.
- **Mecanismo Oculto:** El código detrás de este método (`create`) enmascara una compleja secuencia de pasos: obtener una conexión del pool (`get_conn`), construir una consulta SQL, manejar errores de base de datos (`try/except/rollback`), obtener el ID generado y finalmente cerrar la conexión (`close_conn`).
- **Justificación Académica:** El usuario de la clase DAO (el desarrollador de la capa superior) está **abstracto** de los detalles de la persistencia de datos (tipo de base de datos, manejo de cursos, *pooling* de conexiones). Esta separación estricta entre la lógica de negocio y la lógica de datos cumple con el principio de **Separación de Responsabilidades** (que es un subproducto del buen uso de la Abstracción).

### 4. Polimorfismo (Polymorphism)

El polimorfismo (que significa "muchas formas") se utiliza en su sistema a través de la capacidad de diferentes objetos para responder a la misma llamada de método.

- **Evidencia en el Código:**
  - **Polimorfismo por Subtipo/Sobreescritura:** Las clases `Receptionist` y `Employee` pueden responder a la misma llamada de método, como `createReservation()` (visto en `employee.py` y `receptionist.py`). El sistema puede tratar ambos objetos de manera uniforme como `Employee`, pero en el futuro, el método real ejecutado variará si la subclase (`Receptionist`) proporciona su propia implementación especializada.
  - **Polimorfismo de Inclusión:** Cuando la capa de persistencia (`EmployeeDAO.create`) recibe un objeto `emp: Employee`, este objeto puede ser una instancia de `Employee`, `Receptionist` o `Bellboy`. Sin

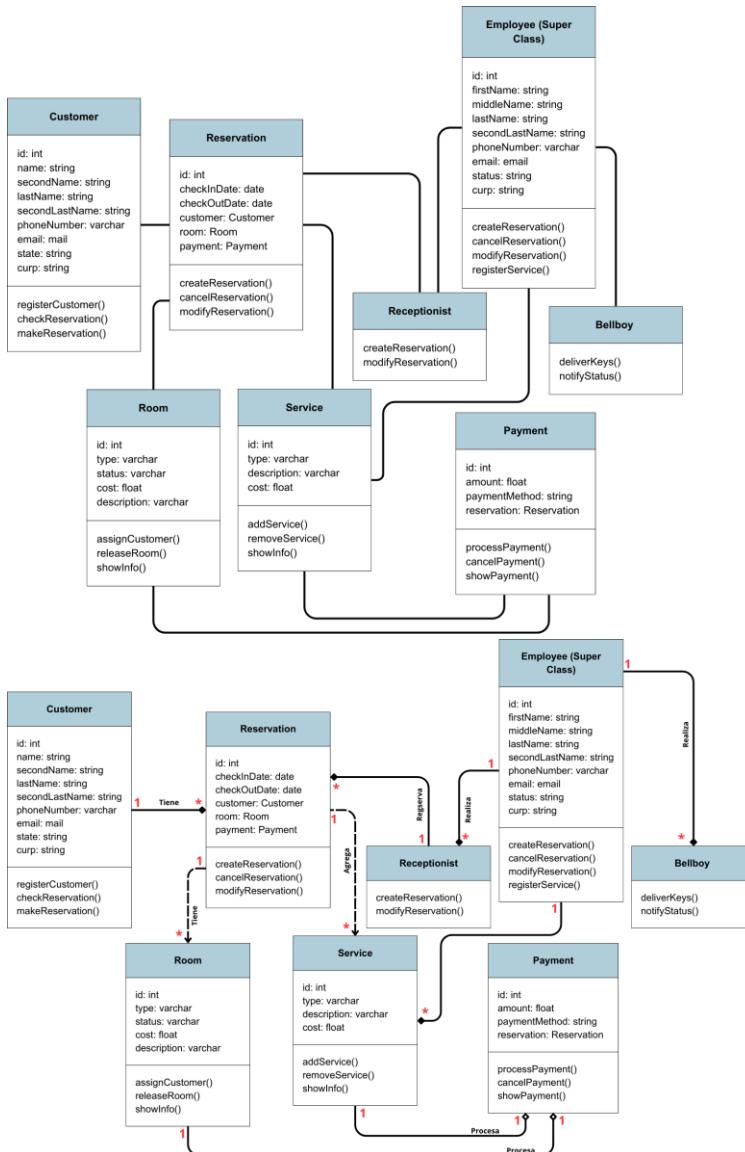
importar el subtipo, la lógica del DAO invoca métodos de la superclase (`emp.getFirstName()`, `emp.getPassword()`) y el código funciona correctamente.

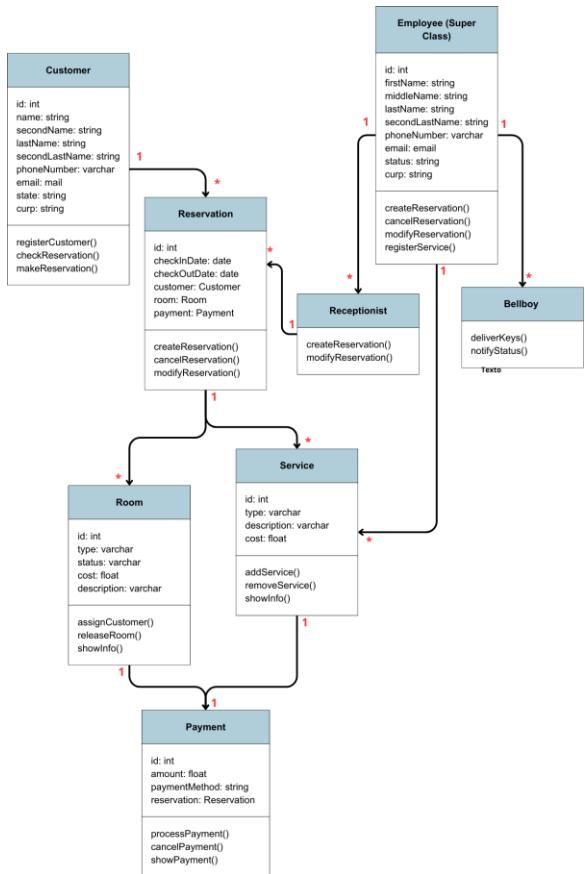
- **Justificación:** El polimorfismo permite un diseño más flexible y extensible. Si se añade un nuevo tipo de empleado (Manager), el resto del código que opera sobre el tipo genérico Employee no tiene que ser modificado, ya que el nuevo objeto se incluirá automáticamente en la interfaz común. Esto es clave para la **extensibilidad** del sistema.

## Modelado UML

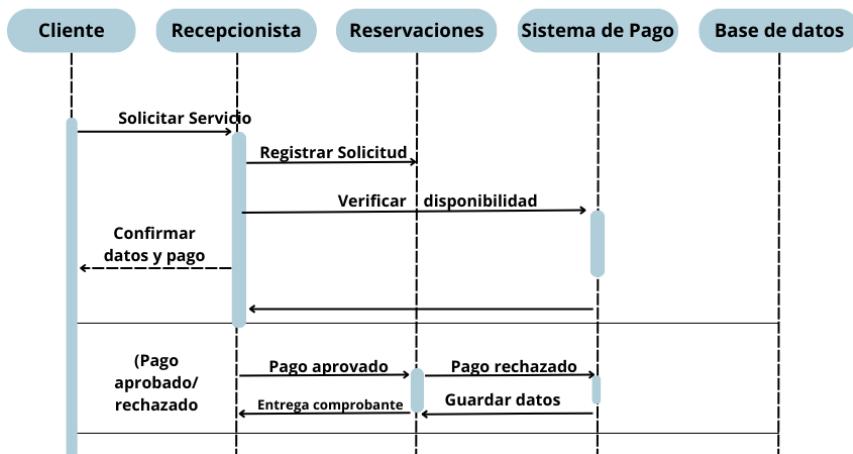
Orden de los diagramas:

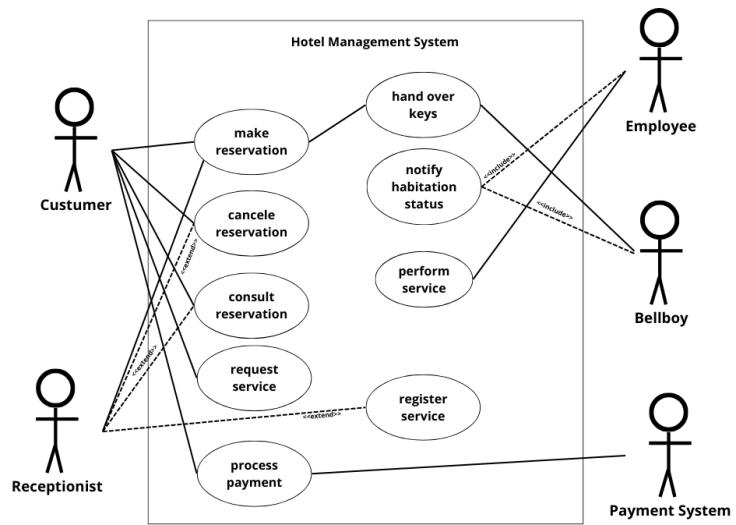
1. Diagrama de clases
2. Asociaciones
3. Agregación y composición
4. Herencia
5. Secuencia
6. Casos de uso
7. Especificación de caso de uso





## Sistema de secuencia





Clase: Customer (Cliente)		Clase: Employee (Empleado – Superclase)	
Caso de uso:	Registrar cliente	Caso de uso:	Administrar operaciones del hotel
Id:	CU-01	Id:	CU-02
Breve descripción:	El recepcionista registra los datos de un nuevo cliente en el sistema	Breve descripción:	Representa las funciones comunes de los empleados, como gestionar reservaciones y servicios.
Actores primarios:	Recepcionista	Actores primarios:	Empleado (Recepcionista o Bellboy)
Actores secundarios:	Cliente	Precondiciones	1. El empleado debe estar registrado y activo. 2. Debe tener los permisos correspondientes según su rol.
Precondiciones	1. El cliente no debe estar previamente registrado. 2. El sistema debe tener acceso a la base de datos.	Flujo principal:	1. El empleado inicia sesión. 2. Accede a las opciones disponibles según su rol. 3. Realiza acciones como crear, modificar o cancelar reservaciones, o registrar servicios. 4. El sistema registra las operaciones realizadas.
Flujo principal:	1. El recepcionista selecciona "Registrar cliente". 2. Ingresa los datos personales del cliente (nombre, CURP, teléfono, correo, etc.). 3. El sistema valida la información. 4. Se guarda el registro del cliente en la base de datos. 5. El sistema confirma el registro exitoso.	Postcondiciones	Se actualiza el registro de actividades del empleado y las operaciones quedan guardadas.
Postcondiciones	El cliente queda registrado y disponible para realizar reservaciones.		
Subclase: Receptionist (Recepcionista)			
Caso de uso:	Crear reserva	Caso de uso:	Entregar llaves y notificar estado
Id:	CU-03	Id:	CU-04
Breve descripción:	El recepcionista genera una nueva reserva para un cliente, asignando habitación y pago.	Breve descripción:	El botones entrega llaves a los huéspedes y actualiza el estado de las habitaciones
Actores primarios:	Recepcionista	Actores primarios:	Botones
Actores secundarios:	Cliente, Sistema de pago	Actores secundarios:	Recepcionista
Precondiciones	1. El cliente debe estar registrado. 2. Debe haber habitaciones disponibles.	Precondiciones	1. El cliente debe tener una reserva activa. 2. La habitación debe estar lista para ocupación.
Flujo principal:	1. El recepcionista busca al cliente. 2. Selecciona habitación, fechas y servicios. 3. El sistema calcula el costo total. 4. Se procesa el pago y se confirma la reserva.	Flujo principal:	1. El botones consulta la habitación asignada al cliente. 2. Entrega las llaves al huésped. 3. Actualiza el estado de la habitación a "ocupada". 4. Notifica al recepcionista que el cliente ingresó.
Postcondiciones	La reserva y el pago quedan registrados y la habitación pasa a estado "reservada".	Postcondiciones	El sistema refleja la habitación como ocupada y se registra la entrega de llaves.

Clase: Reservation (Reservación)		Clase: Room (Habitación)	
Caso de uso:	Gestionar reservación	Caso de uso:	Asignar habitación
Id:	CU-05	Id:	CU-06
Breve descripción:	Permite crear, modificar o cancelar una reservación en el sistema.	Breve descripción:	Asigna una habitación a una reservación y controla su estado.
Actores primarios:	Repcionista	Actores primarios:	Repcionista
Actores secundarios:	Cliente, Sistema de pago	Actores secundarios:	Cliente
Precondiciones	1. El cliente debe estar registrado. 2. Debe haber habitaciones disponibles.	Precondiciones	1. La habitación debe estar disponible.
Flujo principal:	1. Se selecciona el cliente. 2. Se elige habitación y fechas. 3. El sistema calcula el total. 4. Se procesa el pago. 5. El sistema guarda la reservación.	Flujo principal:	1. El recepcionista busca habitaciones disponibles. 2. Asigna una habitación a la reservación. 3. El sistema cambia el estado a "reservada".
Postcondiciones	La reservación queda activa y asociada al cliente y al pago.	Postcondiciones	La habitación queda ocupada o reservada según corresponda.
Clase: Service (Servicio)		Clase: Payment (Pago)	
Caso de uso:	Agregar servicio adicional	Caso de uso:	Procesar pago
Id:	CU-07	Id:	CU-08
Breve descripción:	El recepcionista o el cliente agrega servicios adicionales a una reservación (spa, restaurante, transporte, etc.).	Breve descripción:	Permite registrar, validar y confirmar el pago de una reservación.
Actores primarios:	Repcionista/Cliente	Actores primarios:	Repcionista / Cliente
Precondiciones	1. Debe existir una reservación activa. 2. El servicio debe estar disponible.	Actores secundarios:	Sistema de pago externo
Flujo principal:	1. Se selecciona el servicio deseado. 2. El sistema añade el costo al total de la reservación. 3. Se actualiza la información de la reserva.	Precondiciones	1. Debe existir una reservación generada. 2. El sistema de pago debe estar disponible.
Postcondiciones	El servicio queda registrado y asociado a la reservación del cliente	Flujo principal:	1. El usuario selecciona método de pago (tarjeta, efectivo, etc.). 2. El sistema envía la solicitud al servicio de pago. 3. Se valida la transacción. 4. El sistema confirma el pago y actualiza la reservación.
		Postcondiciones	El pago queda registrado y asociado a la reservación correspondiente.

## Clases

### 1. Customer (Cliente)

Representa a un huésped del hotel.

- **Atributos clave:**

- \_\_id, \_\_name, \_\_lastName, \_\_email, \_\_phone, \_\_curp.

- `__reservations`: Lista para almacenar instancias de `Reservation`.
- `__service_reservations`: Lista para almacenar instancias de `ServiceReservation`.
- **Métodos clave:**
  - `makeReservation(reservation)`: Asocia una reserva de habitación al cliente.
  - `cancelReservation(reservation_id)`: Elimina una reserva de habitación.
  - `makeServiceReservation(service_reservation)`: Asocia una reserva de servicio.
  - `checkReservation()`: Muestra las reservas activas.
  - `Setters y Getters` para todos los atributos.
- **Relación:**
  - **Asociación 1 a M (Uno a Muchos):** Un cliente puede tener **Muchas Reservation** y **Muchas ServiceReservation**.

## 2. Employee (Empleado)

Clase base que representa a todo el personal del hotel.

- **Atributos clave:**
  - `__id`, `__firstName`, `__lastName`, `__email`, `__phone`, `__curp`, `__password`, `__status`.
- **Métodos clave:**
  - `createReservation(reservation)`: Llama al método de creación de la reserva.
  - `cancelReservation(reservation)`: Llama al método de cancelación de la reserva.
  - `modifyReservation(...)`: Llama al método de modificación de la reserva.
  - `registerService(service)`: Agrega un nuevo tipo de servicio al catálogo.
  - `Setters y Getters` para todos los atributos.
- **Relación:**
  - **Herencia:** Es la clase **Padre** (Superclase) de `Receptionist` y `Bellboy` (implícito).

## 3. Receptionist (Repcionista)

Clase especializada para el rol de manejo de *front-desk*.

- **Atributos clave:** Hereda todos los atributos de `Employee`.
- **Métodos clave:**

- `createReservation(reservation)`: Sobreescribe/extiende el comportamiento heredado para las tareas específicas de la recepcionista.
- `modifyReservation(...)`: Sobreescribe/extiende el comportamiento heredado.
- **Relación:**
  - **Herencia:** Receptionist **es un** Employee.

## 4. Reservation (Reserva de Habitación)

Representa la ocupación de una habitación por parte de un cliente.

- **Atributos clave:**
  - `_id`, `_checkIn`, `_checkOut`.
  - `_customer`: Referencia a la instancia de Customer.
  - `_room`: Referencia a la instancia de Room.
  - `_payment`: Referencia opcional a la instancia de Payment.
  - `_services`: Lista de servicios asociados a la reserva.
- **Métodos clave:**
  - `createReservation()`: Asigna la habitación al cliente y cambia el estado de la habitación.
  - `cancelReservation()`: Libera la habitación.
  - `modifyReservation(...)`: Actualiza las fechas de *check-in* o *check-out*.
- **Relación:**
  - **Composición:** Una reserva **contiene** una referencia a un Customer (1 a 1) y una referencia a una Room (1 a 1).
  - **Asociación:** Una reserva puede tener **Muchos Service** (1 a M).

## 5. Payment (Pago)

Representa una transacción financiera asociada a una reserva.

- **Atributos clave:**
  - `_id`, `_amount`, `_paymentMethod`, `_date` (se genera al momento de la creación).
  - `_reservation`: Referencia opcional a la instancia de Reservation.
- **Métodos clave:**
  - `processPayment()`: Simula el procesamiento del pago (actualmente imprime un mensaje).
  - `cancelPayment()`: Simula la cancelación del pago.
- **Relación:**

- **Asociación 1 a 1:** Un pago se relaciona con una única Reservation (gestionada en reservation\_dao.py con link\_payment).

## 6. ServiceReservation (Reserva de Servicio)

Representa la solicitud de un servicio adicional del hotel (p. ej., spa, tour).

- **Atributos clave:**
  - \_\_id, \_\_date\_time.
  - \_\_customer: Referencia a la instancia de Customer.
  - \_\_service: Referencia al tipo de servicio solicitado.
- **Métodos clave:**
  - createReservation(): Asocia la reserva de servicio al historial del cliente.
  - showInfo(): Muestra los detalles de la reserva de servicio.
- **Relación:**
  - **Composición:** Contiene referencias a un Customer y a un Service (1 a 1).

## 7. \*DAO (Clases DAO Genéricas)

Incluye CustomerDAO, EmployeeDAO, ReservationDAO, PaymentDAO, RoomDAO, ServiceDAO.

- **Descripción:** Cada DAO gestiona la persistencia de su respectiva clase de entidad.
- **Atributos clave:** (Generalmente ninguno, operan con la conexión a la base de datos). EmployeeDAO tiene ROLE\_MAPPING.
- **Métodos clave (Comunes a todos):**
  - create(entity): Inserta un nuevo objeto (entity) en la BD.
  - get\_by\_id(id): Recupera un objeto por su identificador.
  - get\_all(): Recupera una lista de todos los objetos.
  - update(...): Modifica el estado de un objeto existente.
  - delete(id): Elimina un registro de la BD.
- **Métodos específicos:**
  - ReservationDAO.link\_payment(res\_id, pay\_id): Método crucial para vincular el pago a la reserva.
  - EmployeeDAO.get\_all(): Utiliza la herencia mediante ROLE\_MAPPING para instanciar el tipo de empleado correcto (Receptionist, Bellboy, etc.) al leer los datos.
- **Relación:**

- **Dependencia:** Cada DAO **depende** de la clase de entidad que gestiona (CustomerDAO depende de Customer) y de la clase de conexión a la BD (db\_connection.py).

## 8. DBConnection

Encargada de gestionar el *pool* de conexiones a la base de datos MySQL.

- **Atributos clave:**
  - dbconfig: Diccionario de configuración tomado de mysql\_env.py.
  - pool: Instancia de MySQLConnectionPool.
- **Métodos clave:**
  - \_\_init\_\_(): Inicializa el *pool* de conexiones.
  - get\_connection(): Proporciona una conexión activa desde el *pool* a las clases DAO.
  - close\_connection(connection): Retorna una conexión al *pool*.
- **Relación:**
  - **Dependencia:** Todas las clases \*DAO **dependen** de las funciones utilitarias de este módulo (get\_conn() y close\_conn()) para interactuar con la BD.

## Códigos

### Bellboy

```
from employee import Employee

class Bellboy(Employee):
    def deliverKeys(self, room):
        pass

    def notifyStatus(self, room):
        print(f"Notifying status of room {room.getId()}: {room.getStatus()}")
```

### Customers

```
from typing import List

class Customer:
```

```
def __init__(self, id, name, secondName, lastName, secondLastName, phone,
email, state, curp, password=""):
    self.__id = id
    self.__name = name
    self.__secondName = secondName
    self.__lastName = lastName
    self.__secondLastName = secondLastName
    self.__phone = phone
    self.__email = email
    self.__state = state
    self.__curp = curp
    self.__password = password
    self.__reservations: List = []
    self.__service_reservations: List = []

def getId(self): return self.__id
def setId(self, id): self.__id = id

def getName(self): return self.__name
def setName(self, name): self.__name = name

def getSecondName(self): return self.__secondName
def setSecondName(self, secondName): self.__secondName = secondName

def getLastNames(self): return self.__lastName
def setLastNames(self, lastName): self.__lastName = lastName

def getSecondLastNames(self): return self.__secondLastName
def setSecondLastNames(self, secondLastName): self.__secondLastName = secondLastName

def getPhone(self): return self.__phone
def setPhone(self, phone): self.__phone = phone

def getEmail(self): return self.__email
def setEmail(self, email): self.__email = email

def getPassword(self): return getattr(self, '_Customer__password', '')
def setPassword(self, password): self.__password = password

def getState(self): return self.__state
def setState(self, state): self.__state = state

def getCurp(self): return self.__curp
def setCurp(self, curp): self.__curp = curp
```

```
def getReservations(self): return self.__reservations

def getServiceReservations(self):
    return self.__service_reservations

def registerCustomer(self):
    print(f"Customer {self.__name} has been registered.")

def makeReservation(self, reservation):
    self.__reservations.append(reservation)
    print(f"Reservation #{reservation.getId()} added to customer {self.__name}'s record.")

def checkReservation(self):
    if not self.__reservations:
        print(f"Customer {self.__name} has no reservations.")
    else:
        print(f"Customer {self.__name} has {len(self.__reservations)} reservation(s).")
        for res in self.__reservations:
            res.showInfo()

def cancelReservation(self, reservation_id):
    initial_count = len(self.__reservations)
    self.__reservations = [res for res in self.__reservations if res.getId() != reservation_id]
    if len(self.__reservations) < initial_count:
        print(f"Reservation #{reservation_id} removed from customer {self.__name}'s record.")
        return True
    return False

def makeServiceReservation(self, service_reservation):
    self.__service_reservations.append(service_reservation)
    print(f"Service Reservation #{service_reservation.getId()} added to customer {self.__name}'s record.")

def cancelServiceReservation(self, reservation_id):
    initial_count = len(self.__service_reservations)
    self.__service_reservations = [res for res in self.__service_reservations if res.getId() != reservation_id]
    if len(self.__service_reservations) < initial_count:
        print(f"Service Reservation #{reservation_id} removed from customer {self.__name}'s record.")
        return True
```

```
    return False
```

## Db\_connection

```
import mysql.connector
from mysql.connector.pooling import MySQLConnectionPool
from mysql_env import HOST, PORT, DATABASE, USER, PASSWORD, POOL_SIZE

class DBConnection:
    def __init__(self):
        try:
            self.dbconfig = {
                "host": HOST,
                "port": int(PORT),
                "database": DATABASE,
                "user": USER,
                "password": PASSWORD,
                "pool_size": POOL_SIZE
            }
            self.pool      = MySQLConnectionPool(pool_name="hotel_pool",
**self.dbconfig)
            print("Pool de conexiones a MySQL creado exitosamente.")

        except mysql.connector.Error as err:
            print(f"Error al conectar con MySQL: {err}")
            raise
        except (ValueError, TypeError) as err:
            print(f"Error en configuracion de conexion: {err}")
            raise

    def get_connection(self):
        return self.pool.get_connection()

    def close_connection(self, connection):
        connection.close()

try:
    _db_connection_instance = DBConnection()

    def get_conn():
        return _db_connection_instance.get_connection()

    def close_conn(connection):
        _db_connection_instance.close_connection(connection)
```

```

except (mysql.connector.Error, ValueError, TypeError) as e:
    print(f"CRITICAL: No se pudo inicializar el pool de conexiones a la BD: {e}")
    raise

if __name__ == '__main__':
    conn_test = get_conn()
    print(f"Conexion de prueba exitosa. ID: {conn_test.connection_id}")
    close_conn(conn_test)

```

## Employee

```

class Employee:
    def __init__(self, id, firstName, secondName, lastName, secondLastName,
phone, email, status, curp, password=""):
        self.__id = id
        self.__firstName = firstName
        self.__secondName = secondName
        self.__lastName = lastName
        self.__secondLastName = secondLastName
        self.__phone = phone
        self.__email = email
        self.__status = status
        self.__curp = curp
        self.__password = password

    def getId(self): return self.__id
    def setId(self, id): self.__id = id

    def getFirstName(self): return self.__firstName
    def setFirstName(self, firstName): self.__firstName = firstName

    def getSecondName(self): return self.__secondName
    def setSecondName(self, secondName): self.__secondName = secondName

    def getLastNames(self): return self.__lastName
    def setLastNames(self, lastName): self.__lastName = lastName

    def getSecondLastName(self): return self.__secondLastName
    def setSecondLastName(self, secondLastName): self.__secondLastName = secondLastName

    def getPhone(self): return self.__phone
    def setPhone(self, phone): self.__phone = phone

```

```

def getEmail(self): return self.__email
def setEmail(self, email): self.__email = email

def getPassword(self): return getattr(self, '_Employee__password', '')
def setPassword(self, password): self.__password = password

def getStatus(self): return self.__status
def setStatus(self, status): self.__status = status

def getCurp(self): return self.__curp
def setCurp(self, curp): self.__curp = curp

def createReservation(self, reservation):
    reservation.createReservation()
    print(f"Employee {self.__firstName} created a reservation for room {reservation.getRoom().getId()}.")

def cancelReservation(self, reservation):
    reservation.cancelReservation()
    print(f"Employee {self.__firstName} cancelled reservation for room {reservation.getRoom().getId()}.")

def modifyReservation(self, reservation, newCheckIn=None, newCheckOut=None):
    reservation.modifyReservation(newCheckIn, newCheckOut)
    print(f"Employee {self.__firstName} modified reservation for room {reservation.getRoom().getId()}.")

def registerService(self, service):
    service.addService()
    print(f"Employee {self.__firstName} registered service {service.getType()}.")

```

## Execute

```

import tkinter as tk
from tkinter import messagebox, ttk
from datetime import date
import re
import uuid

from customer import Customer
from service import Service
from reservationService import ServiceReservation

```

```
from reservation import Reservation
from room import Room
from employee import Employee
from receptionist import Receptionist
from bellboy import Bellboy
from payment import Payment

from dao.employee_dao import EmployeeDAO
from dao.ServiceDAO import ServiceDAO
from dao.customer_dao import CustomerDAO
from dao.room_dao import RoomDAO
from dao.payment_dao import PaymentDAO
from dao.reservation_dao import ReservationDAO

COLOR_PRIMARY = '#1A237E'
COLOR_SECONDARY = '#283593'
COLOR_ACCENT = '#D4AF37'
COLOR_BG = '#F5F6FA'
COLOR_WHITE = '#FFFFFF'
COLOR_TEXT = '#2C3E50'
COLOR_TEXT_LIGHT = '#7F8C8D'
COLOR_DANGER = '#C0392B'

FONT_HEADER = ('Segoe UI', 20, 'bold')
FONT_SUBHEADER = ('Segoe UI', 14, 'bold')
FONT_BODY = ('Segoe UI', 10)
FONT_BODY_BOLD = ('Segoe UI', 10, 'bold')
FONT_BUTTON = ('Segoe UI', 10, 'bold')

#inicializar datos desde la base de datos
def init_data_from_db():
    print("--- Cargando datos iniciales desde la Base de Datos ---")
    customer_dao = CustomerDAO()
    employee_dao = EmployeeDAO()
    service_dao = ServiceDAO()
    room_dao = RoomDAO()
    reservation_dao = ReservationDAO()

    all_customers = customer_dao.get_all()
    all_employees = employee_dao.get_all()
    all_services = service_dao.get_all()
    all_rooms = room_dao.get_all()
    all_reservations = reservation_dao.get_all()

    service_reservations = {}
```

```

        return {
            'customers': {c.getEmail(): c for c in all_customers},
            'employees': {e.getEmail(): e for e in all_employees},
            'rooms': {r.getId(): r for r in all_rooms},
            'reservations': {r.getId(): r for r in all_reservations},
            'service_reservations': service_reservations,
            'services': {s.getId(): s for s in all_services}
        }

    def center_window(window, width, height):
        screen_width = window.winfo_screenwidth()
        screen_height = window.winfo_screenheight()
        x = int((screen_width/2) - (width/2))
        y = int((screen_height/2) - (height/2))
        window.geometry(f"{width}x{height}+{x}+{y}")

    class ModernButton(tk.Button):
        def __init__(self, parent, text, command=None, **kwargs):
            button_type = kwargs.pop('type', kwargs.pop('button_type', 'primary'))

            background = COLOR_PRIMARY if button_type == "primary" else (COLOR_ACCENT if button_type == "accent" else COLOR_WHITE)
            foreground = COLOR_WHITE if button_type in ["primary", "accent"] else COLOR_PRIMARY

            super().__init__(parent, text=text, command=command,
                            bg=background, fg=foreground,
                            font=FONT_BUTTON, relief="flat",
                            activebackground=COLOR_SECONDARY,
                            activeforeground=COLOR_WHITE,
                            bd=0, padx=20, pady=10, cursor="hand2", **kwargs)

    class HotelGUI:
        def __init__(self, master):
            self.master = master
            master.title("Le Villa Hotel Management")
            master.configure(bg=COLOR_BG)
            center_window(master, 900, 600)

            self.style = ttk.Style()
            self.style.theme_use('clam')

            self.style.configure('TFrame', background=COLOR_BG)
            self.style.configure('Card.TFrame', background=COLOR_WHITE,
relief="groove", borderwidth=1)

```

```

        self.style.configure('TLabel',                                     background=COLOR_BG,
foreground=COLOR_TEXT, font=FONT_BODY)
        self.style.configure('Card.TLabel',                               background=COLOR_WHITE,
foreground=COLOR_TEXT, font=FONT_BODY)
        self.style.configure('Header.TLabel',                             font=FONT_HEADER,
foreground=COLOR_PRIMARY, background=COLOR_BG)
        self.style.configure('Title.TLabel',                             font=FONT_SUBHEADER,
foreground=COLOR_PRIMARY, background=COLOR_WHITE)

        self.style.configure('TEntry',          padding=5,           relief="flat",
borderwidth=1)
        self.style.map('TEntry', bordercolor=[('focus', COLOR_PRIMARY)])


self.container = ttk.Frame(master)
self.container.pack(fill="both", expand=True)

self.data = init_data_from_db()

self.next_reservation_id = max(self.data['reservations'].keys()) + 1 if
self.data['reservations'] else 1
self.next_service_reservation_id = max(self.data['service_reservations'].keys()) + 1 if
self.data['service_reservations'] else 1
self.next_payment_id = 1
self.data['payments'] = {}


self.frames = {}

self.employee_dao = EmployeeDAO()
self.service_dao = ServiceDAO()
self.customer_dao = CustomerDAO()
self.room_dao = RoomDAO()
self.payment_dao = PaymentDAO()
self.reservation_dao = ReservationDAO()

for F in (WelcomeScreen, LoginFormScreen, LoginSuccessScreen,
MainMenuScreen):
    page_name = F.__name__
    frame = F(parent=self.container, controller=self)
    self.frames[page_name] = frame
    frame.grid(row=0, column=0, sticky="nsew")

self.show_frame("WelcomeScreen")

def show_frame(self, page_name, user_type=None, user_obj=None,
login_type=None):

```

```

frame = self.frames[page_name]

    if page_name == "LoginSuccessScreen" or page_name == "MainMenuScreen":
        frame.set_user(user_type, user_obj)
    elif page_name == "LoginFormScreen" and login_type:
        frame.set_login_type(login_type)

frame.tkraise()

def add_new_reservation(self, new_reservation):
    res_id = self.next_reservation_id
    self.data['reservations'][res_id] = new_reservation
    self.next_reservation_id += 1
    return res_id

def add_new_service_reservation(self, new_service_reservation):
    res_id = self.next_service_reservation_id
    self.data['service_reservations'][res_id] = new_service_reservation
    self.next_service_reservation_id += 1
    return res_id

def add_new_customer(self, customer_obj):
    self.data['customers'][customer_obj.getEmail()] = customer_obj
    return customer_obj

def add_new_employee(self, employee_obj):
    key = str(employee_obj.getId())
    self.data['employees'][key] = employee_obj
    return employee_obj

def add_new_payment(self, payment_obj):
    pay_id = self.next_payment_id
    self.data['payments'][pay_id] = payment_obj
    self.next_payment_id += 1
    return payment_obj

class CreateServiceReservationWindow(tk.Toplevel):
    def __init__(self, master, controller):
        tk.Toplevel.__init__(self, master)
        self.controller = controller
        self.title("Solicitar Nuevo Servicio")
        center_window(self, 500, 600)
        self.configure(bg=COLOR_BG)

        card = ttk.Frame(self, style='Card.TFrame', padding=20)
        card.pack(fill="both", expand=True, padx=20, pady=20)

```

```

        ttk.Label(card,           text="          Solicitar      Servicio",
style='Title.TLabel').pack(pady=(0, 20))

        form_frame = ttk.Frame(card, style='Card.TFrame')
        form_frame.pack(fill='both', expand=True)

        ttk.Label(form_frame, text="Email del Cliente", style='Card.TLabel',
font=FONT_BODY_BOLD).pack(anchor='w')
        self.email_entry = ttk.Entry(form_frame, width=40)
        self.email_entry.pack(fill='x', pady=(5, 15))

        if controller.frames['MainMenuScreen'].user_type == 'Customer' and
controller.frames['MainMenuScreen'].user_obj:
            self.email_entry.insert(0,
controller.frames['MainMenuScreen'].user_obj.getEmail())
            self.email_entry.config(state='readonly')
        else:
            self.email_entry.insert(0, "andrea@mail.com")

        ttk.Label(form_frame,           text="Habitacion      (Opcional)",
style='Card.TLabel', font=FONT_BODY_BOLD).pack(anchor='w')
        self.room_number_entry = ttk.Entry(form_frame, width=40)
        self.room_number_entry.pack(fill='x', pady=(5, 15))
        self.room_number_entry.insert(0, "101")

        ttk.Label(form_frame,           text="Servicio",       style='Card.TLabel',
font=FONT_BODY_BOLD).pack(anchor='w')
        self.services = self.controller.data.get('services', {})
        service_options = [{"s.getId(): {s.getType()} ${s.getCost():.2f}}"
for s in self.services.values()]

        self.service_var = tk.StringVar(form_frame)
        self.service_combo = ttk.Combobox(form_frame,
textvariable=self.service_var, values=service_options, state="readonly")
        self.service_combo.pack(fill='x', pady=(5, 15))
        if service_options: self.service_combo.set(service_options[0])

        ttk.Label(form_frame,           text="Fecha/Hora",       style='Card.TLabel',
font=FONT_BODY_BOLD).pack(anchor='w')
        self.datetime_entry = ttk.Entry(form_frame, width=40)
        self.datetime_entry.pack(fill='x', pady=(5, 20))
        self.datetime_entry.insert(0, str(date.today()) + " 10:00")

        ModernButton(card,   text="CONFIRMAR SOLICITUD",   type="accent",
command=self.process_service_reservation).pack(fill='x', pady=5)

```

```

        ModernButton(card,           text="CANCELAR",           type="secondary",
command=self.destroy).pack(fill='x', pady=5)

    def validate_datetime(self, dt_str):
        return re.match(r'^\d{4}-\d{2}-\d{2}\s\d{2}:\d{2}$', dt_str)

    def process_service_reservation(self):
        email = self.email_entry.get().strip()
        room_number = self.room_number_entry.get().strip()
        service_info = self.service_var.get().strip()
        date_time_str = self.datetime_entry.get().strip()

        if not all([email, service_info, date_time_str]):
            messagebox.showwarning("Faltan Datos", "Complete todos los
campos.")
            return

        if not self.validate_datetime(date_time_str):
            messagebox.showerror("Error", "Formato de fecha invalido.")
            return

        customer_obj = self.controller.data['customers'].get(email)
        if not customer_obj:
            messagebox.showerror("Error", "Cliente no encontrado.")
            return

        try:
            service_id = int(service_info.split(':')[0])
            service_obj = self.services.get(service_id)
        except:
            return

        new_res_id = self.controller.next_service_reservation_id
        reservation = ServiceReservation(new_res_id, date_time_str,
customer_obj, service_obj)

        if reservation.createReservation():
            self.controller.add_new_service_reservation(reservation)
            PaymentServiceWindow(self.master, self.controller, reservation,
service_obj.getCost(), customer_obj, room_number)
            self.destroy()
        else:
            messagebox.showerror("Error", "No se pudo crear la reserva.")

class PaymentServiceWindow(tk.Toplevel):

```

```

    def __init__(self, master, controller, reservation, service_cost, customer,
room_number):
        tk.Toplevel.__init__(self, master)
        self.controller = controller
        self.reservation = reservation
        self.service_cost = service_cost

        self.title("Pasarela de Pago")
        center_window(self, 450, 550)
        self.configure(bg=COLOR_BG)

        card = ttk.Frame(self, style='Card.TFrame', padding=25)
        card.pack(fill="both", expand=True, padx=20, pady=20)

        ttk.Label(card,           text="Detalles           del           Pago",
style='Title.TLabel').pack(pady=(0, 20))

        info_frame = ttk.Frame(card, style='Card.TFrame')
        info_frame.pack(fill='x', pady=10)

        ttk.Label(info_frame,           text=f"Servicio: {reservation._ServiceReservation_service.getType()}", style='Card.TLabel').pack(anchor='w')
        ttk.Label(info_frame,           text=f"Cliente: {customer.getName()}", style='Card.TLabel').pack(anchor='w')

        ttk.Separator(card, orient='horizontal').pack(fill='x', pady=15)

        ttk.Label(card, text="TOTAL A PAGAR", style='Card.TLabel', font=('Segoe UI', 10)).pack()
        ttk.Label(card,   text=f"${service_cost:.2f}",   style='Card.TLabel', font=('Segoe UI', 24, 'bold'), foreground=COLOR_PRIMARY).pack(pady=5)

        ttk.Label(card,   text="Metodo de Pago:",   style='Card.TLabel', font=FONT_BODY_BOLD).pack(anchor='w', pady=(15, 5))
        self.payment_method_var = tk.StringVar(value="Tarjeta de Credito")

        s = ttk.Style()
        s.configure('TRadiobutton', background=COLOR_WHITE, font=FONT_BODY)

        ttk.Radiobutton(card,           text="Tarjeta de Credito", variable=self.payment_method_var, value="Tarjeta de Credito").pack(anchor='w')
        ttk.Radiobutton(card,           text="Efectivo", variable=self.payment_method_var, value="Efectivo").pack(anchor='w')

```

```

        ModernButton(card,           text="PAGAR      AHORA",           type="accent",
command=self.process_payment).pack(fill='x', pady=(30, 5))
        ModernButton(card,           text="CANCELAR",           type="secondary",
command=self.destroy).pack(fill='x', pady=5)

    def process_payment(self):
        payment = Payment(
            self.controller.next_payment_id,
            self.service_cost,
            self.payment_method_var.get(),
            self.reservation
        )
        payment.processPayment()
        new_payment_id = self.controller.payment_dao.create(payment)
        if new_payment_id:
            messagebox.showinfo("Pago Exitoso", f"El pago #{new_payment_id} ha
sido procesado y guardado.")
        else:
            messagebox.showerror("Error de Base de Datos", "El pago fue
procesado pero no se pudo guardar en la base de datos.")
        self.destroy()

class WelcomeScreen(ttk.Frame):
    def __init__(self, parent, controller):
        ttk.Frame.__init__(self, parent)
        self.controller = controller

        header = tk.Frame(self, bg=COLOR_PRIMARY, height=150)
        header.pack(fill='x')

        title = tk.Label(header, text="LE VILLA", font=('Times New Roman', 40,
'bold'), bg=COLOR_PRIMARY, fg=COLOR_ACCENT)
        title.place(relx=0.5, rely=0.5, anchor='center')

        subtitle = tk.Label(header, text="HOTEL & SPA", font=('Segoe UI', 12,
'bold'), bg=COLOR_PRIMARY, fg=COLOR_WHITE)
        subtitle.place(relx=0.5, rely=0.75, anchor='center')

        content = tk.Frame(self, bg=COLOR_BG)
        content.pack(fill='both', expand=True, padx=50, pady=30)

        tk.Label(content,   text="Seleccione su perfil para acceder",
font=FONT_SUBHEADER, bg=COLOR_BG, fg=COLOR_TEXT).pack(pady=20)

        btn_container = tk.Frame(content, bg=COLOR_BG)
        btn_container.pack()

```

```

        ModernButton(btn_container, text="ACCESO HUESPED", type="primary",
                      command=lambda: controller.show_frame("LoginFormScreen",
login_type="Customer")).grid(row=0, column=0, padx=10, pady=10)

        ModernButton(btn_container, text="ACCESO EMPLEADO", type="primary",
                      command=lambda: controller.show_frame("LoginFormScreen",
login_type="Employee")).grid(row=0, column=1, padx=10, pady=10)

    tk.Frame(content, height=1, bg="#DDDDDD", width=300).pack(pady=20)

    reg_frame = tk.Frame(content, bg=COLOR_BG)
    reg_frame.pack()

    tk.Button(reg_frame, text="Crear cuenta Huesped", font=('Segoe UI', 9,
'underline'),
              bg=COLOR_BG, fg=COLOR_TEXT, bd=0, cursor="hand2",
              command=lambda: RegisterCustomerWindow(controller.master,
controller)).pack(side="left", padx=20)

    tk.Button(reg_frame, text="Crear cuenta Empleado", font=('Segoe UI', 9,
'underline'),
              bg=COLOR_BG, fg=COLOR_TEXT, bd=0, cursor="hand2",
              command=lambda: RegisterEmployeeWindow(controller.master,
controller)).pack(side="left", padx=20)

class LoginFormScreen(ttk.Frame):
    def __init__(self, parent, controller):
        ttk.Frame.__init__(self, parent)
        self.controller = controller

        right_panel = tk.Frame(self, bg=COLOR_BG)
        right_panel.pack(fill='both', expand=True, padx=50, pady=30)

        center_frame = tk.Frame(right_panel, bg=COLOR_BG)
        center_frame.place(relx=0.5, rely=0.5, anchor='center')

        self.title_label = tk.Label(center_frame, text="INICIAR SESION",
font=FONT_HEADER, bg=COLOR_BG, fg=COLOR_PRIMARY)
        self.title_label.pack(pady=(0, 30), anchor='w')

        tk.Label(center_frame, text="Email", font=FONT_BODY_BOLD, bg=COLOR_BG,
fg=COLOR_TEXT).pack(anchor='w')
        self.email_entry = ttk.Entry(center_frame, width=35)

```

```

        self.email_entry.pack(pady=(5, 15))

        tk.Label(center_frame,      text="Contraseña",      font=FONT_BODY_BOLD,
bg=COLOR_BG, fg=COLOR_TEXT).pack(anchor='w')
        self.password_entry = ttk.Entry(center_frame, width=35, show="*")
        self.password_entry.pack(pady=(5, 25))

        ModernButton(center_frame,      text="INGRESAR",      type="accent",
command=self.login).pack(fill='x', pady=5)
        ModernButton(center_frame,      text="VOLVER",      type="secondary",
command=lambda: controller.show_frame("WelcomeScreen")).pack(fill='x', pady=5)

    def set_login_type(self, login_type):
        self.login_type = login_type
        if login_type == "Customer":
            self.title_label.config(text="ACCESO HUESPED")
            self.email_entry.delete(0, tk.END)
        else:
            self.title_label.config(text="ACCESO EMPLEADO")
            self.email_entry.delete(0, tk.END)

    def login(self):
        email = self.email_entry.get().strip()
        password = self.password_entry.get()

        if self.login_type == "Customer":
            user_obj = self.controller.data['customers'].get(email)
            if user_obj and user_obj.getPassword() == password:
                self.controller.show_frame("MainMenuScreen",
user_type="Customer", user_obj=user_obj)
            else:
                messagebox.showerror("Error", "Credenciales inválidas.")
        else:
            user_obj = self.controller.data['employees'].get(email)
            if user_obj and user_obj.getPassword() == password:
                self.controller.show_frame("MainMenuScreen",
user_type="Employee", user_obj=user_obj)
            else:
                messagebox.showerror("Error", "Credenciales invalidadas.")

class LoginSuccessScreen(ttk.Frame):
    def __init__(self, parent, controller):
        ttk.Frame.__init__(self, parent)
class MainMenuScreen(ttk.Frame):
    def __init__(self, parent, controller):

```

```

ttk.Frame.__init__(self, parent)
self.controller = controller

top_bar = tk.Frame(self, bg=COLOR_WHITE, height=60)
top_bar.pack(side='top', fill='x')
top_bar.pack_propagate(False)

tk.Label(top_bar, text="LE VILLA -HOTEL & SPA-", font=('Times New Roman', 14, 'bold'), bg=COLOR_WHITE, fg=COLOR_PRIMARY).pack(side='left', padx=20)
self.user_label = tk.Label(top_bar, text="", font=FONT_BODY, bg=COLOR_WHITE, fg=COLOR_TEXT)
self.user_label.pack(side='right', padx=20)

self.main_area = tk.Frame(self, bg=COLOR_BG)
self.main_area.pack(fill='both', expand=True, padx=40, pady=40)

self.grid_buttons_frame = tk.Frame(self.main_area, bg=COLOR_BG)
self.grid_buttons_frame.pack(anchor='center')

def create_dashboard_card(self, parent, icon, title, command, row, col):
frame = tk.Frame(parent, bg=COLOR_WHITE, width=200, height=150)
frame.grid(row=row, column=col, padx=15, pady=15)
frame.pack_propagate(False)

def on_click(e): command()

icon_lbl = tk.Label(frame, text=icon, font=('Segoe UI', 30), bg=COLOR_WHITE, fg=COLOR_PRIMARY, cursor="hand2")
icon_lbl.pack(expand=True)
icon_lbl.bind("<Button-1>", on_click)

txt_lbl = tk.Label(frame, text=title, font=('Segoe UI', 11, 'bold'), bg=COLOR_WHITE, fg=COLOR_TEXT, cursor="hand2")
txt_lbl.pack(pady=(0, 20))
txt_lbl.bind("<Button-1>", on_click)

frame.bind("<Button-1>", on_click)

def set_user(self, user_type, user_obj):
self.user_type = user_type
self.user_obj = user_obj

role_text = "Huésped" if user_type == "Customer" else "Empleado"

```

```

        name = user_obj.getName() if user_type == "Customer" else
user_obj.getFirstName()
        self.user_label.config(text=f"{role_text}: {name} | ⚙ Cerrar Sesión")
        self.user_label.bind("<Button-1>", lambda e:
self.controller.show_frame("WelcomeScreen"))
        self.user_label.configure(cursor="hand2")

    for widget in self.grid_buttons_frame.winfo_children():
        widget.destroy()

        self.create_dashboard_card(self.grid_buttons_frame, "🕒", "Reservar
Habitacion", self.open_create_reservation, 0, 0)
        self.create_dashboard_card(self.grid_buttons_frame, "🛎", "Solicitar
Servicio", self.open_create_service_reservation, 0, 1)
        self.create_dashboard_card(self.grid_buttons_frame, "⌚", "Ver mis
Reservas", self.view_room_reservation, 0, 2)

        self.create_dashboard_card(self.grid_buttons_frame, "💡", "Ver
Servicios Activos", self.view_service_reservation, 1, 0)
        self.create_dashboard_card(self.grid_buttons_frame, "ℹ️", "Info de las
habitaciones", self.view_room_info, 1, 1)

    def open_create_reservation(self):
CreateReservationWindow(self.controller.master, self.controller)
    def open_create_service_reservation(self):
CreateServiceReservationWindow(self.controller.master, self.controller)

def view_room_reservation(self):
    ViewReservationsWindow(self.controller.master, self.controller)

def view_service_reservation(self):
    ViewServicesWindow(self.controller.master, self.controller)

def view_room_info(self):
    RoomInfoWindow(self.controller.master, self.controller)

class CreateReservationWindow(tk.Toplevel):
    def __init__(self, master, controller):
        tk.Toplevel.__init__(self, master)
        self.controller = controller

        self.title("Reservar Habitacion") # <-- El código huérfano empieza aquí
        center_window(self, 600, 700)

```

```

    self.configure(bg=COLOR_BG)

    container = ttk.Frame(self, style='Card.TFrame', padding=20)
    container.pack(fill='both', expand=True, padx=20, pady=20)

        ttk.Label(container, text="Nueva Reserva",
style='Title.TLabel').pack(pady=(0, 20))

        f = ttk.Frame(container, style='Card.TFrame')
        f.pack(fill='both', expand=True)

        self.create_field(f, "Email Cliente:", 0)
        self.email_entry = ttk.Entry(f, width=30); self.email_entry.grid(row=0,
column=1, pady=10)
        if controller.frames['MainMenuScreen'].user_type == 'Customer':
            self.email_entry.insert(0,
controller.frames['MainMenuScreen'].user_obj.getEmail())

        self.create_field(f, "Tipo Habitacion:", 1)
        self.room_type_var = tk.StringVar(value="Suite")
        self.cb_type = ttk.Combobox(f, textvariable=self.room_type_var,
values=["Suite", "Doble", "Individual"], state="readonly")
        self.cb_type.grid(row=1, column=1, pady=10)

        self.create_field(f, "Registro entrada (AAAA-MM-DD):", 2)
        self.in_entry = ttk.Entry(f); self.in_entry.insert(0,
str(date.today())); self.in_entry.grid(row=2, column=1, pady=10)

        self.create_field(f, "Registro salida (AAAA-MM-DD):", 3)
        self.out_entry = ttk.Entry(f); self.out_entry.insert(0, "2025-12-31");
self.out_entry.grid(row=3, column=1, pady=10)

        self.create_field(f, "Habitacion Disponible:", 4)
        all_rooms = self.controller.data.get('rooms', {}).values()
        available_rooms = [r for r in all_rooms if r.getStatus().lower() ==
'available']
        room_options = [f'{r.getId()}: {r.getType()} (${r.getCost():.2f})' for
r in available_rooms]
        self.room_var = tk.StringVar()
        self.room_cb = ttk.Combobox(f, textvariable=self.room_var,
values=room_options, state='readonly')
        self.room_cb.grid(row=4, column=1, pady=10)
        if room_options: self.room_cb.set(room_options[0])

        ttk.Label(f, text="Servicio Adicional:", style='Card.TLabel',
font=FONT_BODY_BOLD).grid(row=5, column=0, sticky='w', padx=10)

```

```

        services = self.controller.data.get('services', {})
        service_options = ["Ninguno"] + [{"${s.getId()}": ${s.getType()}}]
        (${s.getCost():.2f})" for s in services.values()]
        self.service_var = tk.StringVar(value="Ninguno")
        self.service_cb = ttk.Combobox(f, textvariable=self.service_var,
values=service_options, state='readonly')
        self.service_cb.grid(row=5, column=1, pady=10)

        ModernButton(container, text="CONFIRMAR RESERVA", type="primary",
command=self.process).pack(fill='x', pady=(10,8))
        ModernButton(container, text="REGRESAR", type="secondary",
command=self.destroy).pack(fill='x', pady=(0,10))

    def create_field(self, parent, text, row):
        ttk.Label(parent, text=text, style='Card.TLabel',
font=FONT_BODY_BOLD).grid(row=row, column=0, sticky='w', padx=10)

    def process(self):
        room_info = self.room_var.get()
        try:
            rid = int(room_info.split(':')[0])
        except Exception:
            messagebox.showerror("Error", "Debe seleccionar una habitacion valida.")
        return

        room = self.controller.data.get('rooms', {}).get(rid)
        cust = self.controller.data['customers'].get(self.email_entry.get())

        if not room or not cust:
            messagebox.showerror("Error", "Datos invalidos: cliente o habitacion no encontrados.")
        return

        from datetime import datetime
        try:
            check_in = datetime.strptime(self.in_entry.get().strip(), "%Y-%m-%d").date()
            check_out = datetime.strptime(self.out_entry.get().strip(), "%Y-%m-%d").date()
        except Exception:
            messagebox.showerror("Fecha invalida", "Formato de fecha invalido.
Use AAAA-MM-DD.")
        return

        if check_out <= check_in:

```

```

        messagebox.showerror("Fechas invalidas", "La fecha de salida debe
ser posterior a la fecha de entrada.")
        return

    try:
        res = Reservation(self.controller.next_reservation_id,
self.in_entry.get().strip(), self.out_entry.get().strip(), cust, room)
        if not res.createReservation():
            messagebox.showerror("Error", "Habitacion no disponible.")
            return

        num_nights = (check_out - check_in).days
        room_subtotal = room.getCost() * num_nights

        additional_service = None
        service_subtotal = 0
        sel = self.service_var.get()
        svc_dict = self.controller.data.get('services', {})
        if sel and sel != "Ninguno":
            try:
                sid = int(sel.split(':')[0])
                additional_service = svc_dict.get(sid)
                if additional_service:
                    service_subtotal = additional_service.getCost()
            except Exception:
                additional_service = None

        total_cost = room_subtotal + service_subtotal

        new_res_id = self.controller.reservation_dao.create(res,
total_cost)
        if not new_res_id:
            messagebox.showerror("Error de Base de Datos", "No se pudo
registrar la reserva. Intente de nuevo.")
            return

        self.controller.data['reservations'][new_res_id] = res

        PaymentWindow(self.master, self.controller, res, total_cost, cust,
additional_service, room_subtotal, service_subtotal)
        self.destroy()
    except Exception:
        messagebox.showerror("Error", "Error procesando datos.")

class PaymentWindow(tk.Toplevel):

```

```

    def __init__(self, master, controller, reservation, total_cost, customer,
additional_service=None, room_subtotal=None, service_subtotal=0):
        tk.Toplevel.__init__(self, master)
        self.controller = controller
        self.reservation = reservation
        self.total_cost = total_cost
        self.room_subtotal = room_subtotal
        self.service_subtotal = service_subtotal
        self.additional_service = additional_service

        self.title("Check-out y Pago")
        center_window(self, 400, 420)
        self.configure(bg=COLOR_BG)

    card = ttk.Frame(self, style='Card.TFrame', padding=20)
    card.pack(fill='both', expand=True, padx=20, pady=20)

    ttk.Label(card,           text="Pago           de           Habitacion",
style='Title.TLabel').pack(pady=10)

    if self.room_subtotal is None:
        try:
            from datetime import datetime
            check_in   = datetime.strptime(self.reservation.getCheckIn(),
"%Y-%m-%d").date()
            check_out  = datetime.strptime(self.reservation.getCheckOut(),
"%Y-%m-%d").date()
            nights = (check_out - check_in).days
            self.room_subtotal = self.reservation.getRoom().getCost() *
max(1, nights)
        except Exception:
            self.room_subtotal = self.total_cost

    ttk.Label(card,      text=f"Habitacion: ${self.room_subtotal:.2f}",
style='Card.TLabel').pack(pady=(8,2))
    if self.additional_service and self.service_subtotal:
        try:
            svc_name = self.additional_service.getType()
        except Exception:
            svc_name = 'Servicio'
        ttk.Label(card,  text=f"{svc_name}: ${self.service_subtotal:.2f}",
style='Card.TLabel').pack(pady=(0,4))

    ttk.Label(card,  text=f"Total: ${self.total_cost:.2f}", font=('Segoe
UI', 20, 'bold'), foreground=COLOR_PRIMARY,
background=COLOR_WHITE).pack(pady=12)

```

```

        self.method = tk.StringVar(value="Tarjeta")
        ttk.Radiobutton(card,      text="Tarjeta",      variable=self.method,
value="Tarjeta").pack()
        ttk.Radiobutton(card,      text="Efectivo",    variable=self.method,
value="Efectivo").pack()

        btn_frame = ttk.Frame(card)
        btn_frame.pack(fill='x', pady=(12,0))
        ModernButton(btn_frame,                      text="PAGAR",
command=self.pay).pack(side='left', fill='x', expand=True, padx=(0,6))
        ModernButton(btn_frame,          text="REGRESAR", type='secondary',
command=self.destroy).pack(side='right', fill='x', expand=True, padx=(6,0))

    def pay(self):
        p = Payment(self.controller.next_payment_id,   self.total_cost,
self.method.get(), self.reservation)
        p.processPayment()
        new_payment_id = self.controller.payment_dao.create(p)
        if new_payment_id:
            messagebox.showinfo("Exito", f"Pago #{new_payment_id} completado y
guardado.")

        self.controller.reservation_dao.link_payment(self.reservation.getId(),
new_payment_id)
        else:
            messagebox.showerror("Error de Base de Datos", "El pago fue
procesado pero no se pudo guardar en la base de datos.")
        self.destroy()

class ViewReservationsWindow(tk.Toplevel):
    def __init__(self, master, controller):
        tk.Toplevel.__init__(self, master)
        self.controller = controller
        self.title("Reservas")
        center_window(self, 600, 420)
        self.configure(bg=COLOR_BG)

        card = ttk.Frame(self, style='Card.TFrame', padding=20)
        card.pack(fill='both', expand=True, padx=20, pady=20)
        ttk.Label(card,                                     text="Reservas",
style='Title TLabel').pack(pady=(0,10))

        tree_frame = ttk.Frame(card)
        tree_frame.pack(fill='both', expand=True, pady=5)

```

```

columns = ('id', 'customer', 'room', 'check_in', 'check_out', 'cost')
self.tree = ttk.Treeview(tree_frame, columns=columns, show='headings')

self.tree.heading('id', text='ID')
self.tree.heading('customer', text='Cliente')
self.tree.heading('room', text='Habitación')
self.tree.heading('check_in', text='Check-In')
self.tree.heading('check_out', text='Check-Out')
self.tree.heading('cost', text='Costo Total')

self.tree.column('id', width=50, anchor='center')
self.tree.column('customer', width=150)
self.tree.column('room', width=80, anchor='center')
self.tree.column('check_in', width=100, anchor='center')
self.tree.column('check_out', width=100, anchor='center')
self.tree.column('cost', width=80, anchor='e')

self.tree.pack(side='left', fill='both', expand=True)

scrollbar      =      ttk.Scrollbar(tree_frame,          orient='vertical',
command=self.tree.yview)
self.tree.configure(yscrollcommand=scrollbar.set)
scrollbar.pack(side='right', fill='y')

self.load_reservations()

btn_frame = ttk.Frame(card, style='Card.TFrame')
btn_frame.pack(fill='x', pady=(10, 0))

if self.controller.frames['MainMenuScreen'].user_type == 'Employee':
    ModernButton(btn_frame, text="Eliminar Reserva Seleccionada",
type="secondary", command=self.delete_selected_reservation).pack(side='left')

    ModernButton(btn_frame,                                         text="Cerrar",
command=self.destroy).pack(side='right')

def load_reservations(self):
    for i in self.tree.get_children():
        self.tree.delete(i)

reservations = self.controller.data.get('reservations', {}).values()
for res in sorted(reservations, key=lambda r: r.getId()):
    customer_name = res.getCustomer().getName()
    room_number = res.getRoom().getRoomNumber()

```

```

        total_cost = f"${res.getRoom().getCost():.2f}" # Simplificado, el
costo real está en la tabla RESERVATIONS
        self.tree.insert('', 'end', values=(res.getId(), customer_name,
room_number, res.getCheckIn(), res.getCheckOut(), total_cost))

    def delete_selected_reservation(self):
        selected_item = self.tree.focus()
        if not selected_item:
            messagebox.showwarning("Sin Selección", "Por favor, seleccione una
reserva de la lista para eliminar.")
            return

        values = self.tree.item(selected_item, 'values')
        reservation_id = int(values[0])
        customer_name = values[1]

        if messagebox.askyesno("Confirmar Eliminación", f"¿Está seguro de que
desea eliminar la reserva #{reservation_id} de {customer_name}?"):
            if self.controller.reservation_dao.delete(reservation_id):
                messagebox.showinfo("Exito", f"La reserva #{reservation_id} ha
sido eliminada.")
                self.tree.delete(selected_item)
                del self.controller.data['reservations'][reservation_id]
            else:
                messagebox.showerror("Error de Base de Datos", "No se pudo
eliminar la reserva.")

class ViewServicesWindow(tk.Toplevel):
    def __init__(self, master, controller):
        tk.Toplevel.__init__(self, master)
        self.controller = controller
        self.title("Servicios Activos")
        center_window(self, 600, 420)
        self.configure(bg=COLOR_BG)

        card = ttk.Frame(self, style='Card.TFrame', padding=20)
        card.pack(fill='both', expand=True, padx=20, pady=20)
        ttk.Label(card, text="Servicios Activos",
style='Title.TLabel').pack(pady=(0,10))

        text_frame = ttk.Frame(card, style='Card.TFrame')
        text_frame.pack(fill='both', expand=True)

        txt = tk.Text(text_frame, wrap='word', bg=COLOR_WHITE, fg=COLOR_TEXT)
        txt.pack(side='left', fill='both', expand=True)

```

```

sb = ttk.Scrollbar(text_frame, orient='vertical', command=txt.yview)
sb.pack(side='right', fill='y')
txt.configure(yscrollcommand=sb.set)

        if controller.frames.get('MainMenuScreen') and
controller.frames['MainMenuScreen'].user_type == 'Employee':
            services = list(controller.data.get('service_reservations',
{}).values())
        else:
            user = controller.frames.get('MainMenuScreen') and
controller.frames['MainMenuScreen'].user_obj
            services = user.getServiceReservations() if user else []

        if not services:
            txt.insert('end', 'No hay servicios activos para mostrar.')
        else:
            for s in services:
                try:
                    txt.insert('end', s.showInfo() + '\n' + ('-'*60) + '\n')
                except Exception:
                    txt.insert('end', str(s) + '\n' + ('-'*60) + '\n')

        txt.config(state='disabled')

        ModernButton(card, text="REGRESAR", type='secondary',
command=self.destroy).pack(fill='x', pady=10)
    
```

```

class RoomInfoWindow(tk.Toplevel):
    def __init__(self, master, controller):
        tk.Toplevel.__init__(self, master)
        self.controller = controller
        self.title("Informacion de Habitaciones")
        center_window(self, 600, 420)
        self.configure(bg=COLOR_BG)

        card = ttk.Frame(self, style='Card.TFrame', padding=20)
        card.pack(fill='both', expand=True, padx=20, pady=20)
        ttk.Label(card, text="Información de Habitaciones",
style='Title.TLabel').pack(pady=(0,10))

        tree_frame = ttk.Frame(card)
        tree_frame.pack(fill='both', expand=True)

        txt = tk.Text(tree_frame, wrap='none', bg=COLOR_WHITE, fg=COLOR_TEXT)
        txt.pack(side='left', fill='both', expand=True)
    
```

```

sb = ttk.Scrollbar(tree_frame, orient='vertical', command=txt.yview)
sb.pack(side='right', fill='y')
txt.configure(yscrollcommand=sb.set)

rooms = controller.data.get('rooms', {}).values()
if not rooms:
    txt.insert('end', 'No hay informacion de habitaciones.')
else:
    for r in sorted(rooms, key=lambda x: x.getId()):
        info = f"ID: {r.getId()} | Tipo: {r.getType()} | Precio: ${r.getCost():.2f} | Estado: {r.getStatus()}\n"
        txt.insert('end', info)

txt.config(state='disabled')

ModernButton(card, text="REGRESAR", type='secondary',
command=self.destroy).pack(fill='x', pady=10)

class RegisterCustomerWindow(tk.Toplevel):
    def __init__(self, master, controller):
        tk.Toplevel.__init__(self, master)
        self.controller = controller
        self.title("Registro Cliente")
        center_window(self, 500, 600)
        self.configure(bg=COLOR_BG)

        c = ttk.Frame(self, style='Card.TFrame', padding=20);
c.pack(fill='both', expand=True, padx=20, pady=20)
        ttk.Label(c, text="Nuevo Huesped", style='Title.TLabel').pack()

        f = ttk.Frame(c, style='Card.TFrame'); f.pack(pady=10)

        self.entries = {}
        fields = ["first_name", "last_name", "email", "phone", "state", "curp"]
        for i, field in enumerate(fields):
            ttk.Label(f, text=field.capitalize(),
style='Card.TLabel').grid(row=i, column=0, sticky='w')
            e = ttk.Entry(f); e.grid(row=i, column=1, pady=5);
        self.entries[field] = e

        ttk.Label(f, text="Password",
style='Card.TLabel').grid(row=6, column=0)
        self.pw = ttk.Entry(f, show="*"); self.pw.grid(row=6, column=1, pady=5)

```

```

        ModernButton(c,    text="REGISTRAR",    command=self.save).pack(fill='x',
pady=(10,8))
        ModernButton(c,              text="REGRESAR",           type='secondary',
command=self.destroy).pack(fill='x',  pady=(0,10))

    def save(self):
        data = {k: v.get().strip() for k, v in self.entries.items()}
        password = self.pw.get().strip()

        if not all([data.get('first_name'),      data.get('last_name'),
data.get('email'), password]):
            messagebox.showwarning("Datos Incompletos", "Nombre, apellido,
email y contraseña son obligatorios.")
            return

        new_customer = Customer(0, data['first_name'], "", data['last_name'],
"", data['phone'], data['email'], data['state'], data['curp'], password)

        new_id = self.controller.customer_dao.create(new_customer)
        if new_id:
            messagebox.showinfo("Registro Exitoso", f"Cliente
{new_customer.getName()} registrado con ID: {new_id}.")
            self.controller.data['customers'][new_customer.getEmail()] =
new_customer
            self.destroy()
        else:
            messagebox.showerror("Error de Base de Datos", "No se pudo registrar
al cliente. Revise la consola.")

    class RegisterEmployeeWindow(tk.Toplevel):
        def __init__(self, master, controller):
            tk.Toplevel.__init__(self, master)
            self.controller = controller
            self.title("Registro Empleado")
            center_window(self, 500, 650)
            self.configure(bg=COLOR_BG)

            c = ttk.Frame(self, style='Card.TFrame', padding=20)
            c.pack(fill='both', expand=True, padx=20, pady=20)
            ttk.Label(c, text="Nuevo Empleado", style='Title TLabel').pack()

            f = ttk.Frame(c, style='Card.TFrame')
            f.pack(pady=10)

            self.entries = {}

```

```

        fields = ["first_name", "middle_name", "last_name", "second_last_name",
"phone", "email", "status", "curp"]
        for i, field in enumerate(fields):
            ttk.Label(f,      text=field.replace('_', ' ').capitalize(),
style='Card.TLabel').grid(row=i, column=0, sticky='w')
            e = ttk.Entry(f)
            e.grid(row=i, column=1, pady=5)
            self.entries[field] = e

            ttk.Label(f,    text="Role",   style='Card.TLabel').grid(row=len(fields),
column=0, sticky='w')
            self.role_var = tk.StringVar(value='Repcionista')
            role_cb      =      ttk.Combobox(f,      textvariable=self.role_var,
values=["Repcionista", "Botones", "Servicio"], state='readonly')
            role_cb.grid(row=len(fields), column=1, pady=5)

            ttk.Label(f,                               text="Password",
style='Card.TLabel').grid(row=len(fields)+1, column=0, sticky='w')
            self.pw   =  ttk.Entry(f,  show='*');  self.pw.grid(row=len(fields)+1,
column=1, pady=5)
            ttk.Label(f,           text="Confirmar",          text="Password",
style='Card.TLabel').grid(row=len(fields)+2, column=0, sticky='w')
            self.pw_confirm       =      ttk.Entry(f,           show='*');
            self.pw_confirm.grid(row=len(fields)+2, column=1, pady=5)

            ModernButton(c,           text="REGISTRAR",          EMPLEADO",
command=self.process_registration).pack(fill='x', pady=(10,8))
            ModernButton(c,           text="REGRESAR",           type="secondary",
command=self.go_back).pack(fill='x', pady=(0,10))

    def process_registration(self):
        data = {k: v.get().strip() for k, v in self.entries.items()}
        password = self.pw.get().strip()
        confirm = self.pw_confirm.get().strip()

        required = ['first_name', 'last_name', 'phone', 'email', 'status',
'curp']
        if not all(data.get(k) for k in required):
            messagebox.showwarning("Datos Incompletos", "Por favor complete los
campos obligatorios.")
            return
        if password == "" or password != confirm:
            messagebox.showerror("Contraseña", "Las contraseñas están vacias o
no coinciden.")
            return

```

```

        try:
            existing_ids = [int(k) for k in self.controller.data['employees'].keys()]
            new_id = max(existing_ids) + 1 if existing_ids else 1
        except Exception:
            new_id = 1

        role = self.role_var.get()
        if role == 'Repcionista':
            new_emp = Receptionist(new_id, data['first_name'],
data.get('middle_name', ''), data['last_name'],
data.get('second_last_name', ''), data['phone'], data['email'], data['status'],
data['curp'], password)
        elif role == 'Botones':
            new_emp = Bellboy(new_id, data['first_name'],
data.get('middle_name', ''), data['last_name'],
data.get('second_last_name', ''), data['phone'], data['email'], data['status'],
data['curp'], password)
        else:
            new_emp = Employee(new_id, data['first_name'],
data.get('middle_name', ''), data['last_name'],
data.get('second_last_name', ''), data['phone'], data['email'], data['status'],
data['curp'], password)

        new_id_from_db = self.controller.employee_dao.create(new_emp)

        if new_id_from_db:
            messagebox.showinfo("Registro Exitoso", f"Empleado {new_emp.getFirstName()} registrado en la base de datos con ID: {new_id_from_db}.")
            self.controller.data['employees'][new_emp.getEmail()] = new_emp
            self.destroy()
        else:
            messagebox.showerror("Error de Base de Datos", "No se pudo registrar al empleado. Revise la consola para más detalles.")

    def go_back(self):
        try:
            self.controller.show_frame("WelcomeScreen")
        except Exception:
            pass
        self.destroy()

if __name__ == '__main__':
    raiz = tk.Tk()

```

```

try:
    from ctypes import windll
    windll.shcore.SetProcessDpiAwareness(1)
except:
    pass
aplicacion = HotelGUI(raiz)
raiz.mainloop()

```

## Mysql\_env

```

from dotenv import load_dotenv
import os

load_dotenv()

HOST = os.getenv("DB_HOST", "localhost")
PORT = int(os.getenv("DB_PORT", "3306"))
DATABASE = os.getenv("DB_NAME", "hotel_le_villa")
USER = os.getenv("DB_USER", "root")
PASSWORD = os.getenv("DB_PASSWORD", "")
POOL_SIZE = int(os.getenv("POOL_SIZE", "5"))

if not HOST or not DATABASE or not USER:
    raise ValueError("Variables de entorno criticas no configuradas: DB_HOST, DB_NAME, DB_USER son obligatorias.")

```

## Payment

```

from datetime import date

class Payment:
    def __init__(self, id, amount, paymentMethod, reservation=None):
        self.__id = id
        self.__amount = amount
        self.__paymentMethod = paymentMethod
        self.__reservation = reservation
        self.__date = date.today()

    def getId(self): return self.__id
    def setId(self, id): self.__id = id

    def getAmount(self): return self.__amount
    def setAmount(self, amount): self.__amount = amount

```

```

def getPaymentMethod(self): return self.__paymentMethod
def setPaymentMethod(self, method): self.__paymentMethod = method

def getReservation(self): return self.__reservation
def setReservation(self, reservation): self.__reservation = reservation

def getDate(self): return self.__date

def processPayment(self):
    print(f"Payment      processed      for      ${self.__amount}      by
{self.__paymentMethod} on {self.__date}.")"

def cancelPayment(self):
    print(f"Payment #{self.__id} canceled.")

def showPayment(self):
    print(f"Payment      #{self.__id}:      ${self.__amount},      method:
{self.__paymentMethod}, date: {self.__date}")
```

## Populate\_db

```

#codigo con empleados de ejemplo para comenzar la BD
def clear_tables(daos):
    print("--- Limpieza de tablas omitida) ---")

def populate_employees(employee_dao):
    from receptionist import Receptionist
    from bellboy import Bellboy
    print("\n--- Creando Empleados ---")
    receptionist = Receptionist(0, "Brigitte", "", "Herrera", "Rodriguez",
"4421111111", "brigitte@mail.com", "Activo", "HERB001122QRO", "brigitte123")
    bellboy = Bellboy(0, "Carlos", "", "Gomez", "Solis", "4422222222",
"carlos@mail.com", "Activo", "GOSC001122QRO", "carlos123")

    if employee_dao.create(receptionist):
        print(f"-> Empleado {receptionist.getFirstName()} creado con ID:
{receptionist.getId()}")
    if employee_dao.create(bellboy):
        print(f"-> Empleado {bellboy.getFirstName()} creado con ID:
{bellboy.getId()}")

def populate_customers(customer_dao):
    from customer import Customer
    print("\n--- Creando Clientes ---")
```

```

        customer1 = Customer(0, "Andrea", "Sarahi", "Lopez", "Guerrero",
"4420000000", "andrea@mail.com", "Querétaro", "LOGA001122QRO", "andrea123")
        customer2 = Customer(0, "Juan", "", "Perez", "Vera", "4423333333",
"juan@mail.com", "Querétaro", "PEVJ001122QRO", "juan123")

    if customer_dao.create(customer1):
        print(f"-> Cliente '{customer1.getName()}' creado con ID:
{customer1.getId()}")
    if customer_dao.create(customer2):
        print(f"-> Cliente '{customer2.getName()}' creado con ID:
{customer2.getId()}")

def populate_services(service_dao):
    from service import Service
    print("\n--- Creando Servicios ---")
    services_to_create = [
        Service(0, "Room Service 24H", 15.00, "Entrega de alimentos y bebidas
a la habitacion a cualquier hora."),
        Service(0, "Lavanderia Express", 30.00, "Lavado, secado y planchado con
entrega en 4 horas."),
        Service(0, "Acceso a Spa", 50.00, "Acceso a sauna, baño turco y piscina
climatizada por dia."),
        Service(0, "Masaje Terapeutico", 85.00, "Sesion privada de masaje de
cuerpo completo (60 minutos)."),
        Service(0, "Cama Adicional", 25.00, "Instalacion de una cama supletoria
en la habitacion."),
        Service(0, "Parqueo con Valet", 10.00, "Servicio de aparcacoches y
recogida del vehiculo.")
    ]
    for svc in services_to_create:
        if service_dao.create(svc):
            print(f"-> Servicio '{svc.getType()}' creado con ID:
{svc.getId()}")

def populate_rooms(room_dao):
    """Puebla la tabla de habitaciones."""
    from room import Room
    print("\n--- Creando Habitaciones ---")
    rooms_to_create = [
        Room(0, "101", "Sencilla", "Available", 120.00, "Una cama Queen size,
ideal para un viajero."),
        Room(0, "205", "Doble", "Available", 180.00, "Dos camas Queen size,
para hasta 4 huéspedes."),
        Room(0, "310", "King Size", "Not available", 220.00, "Una cama King
size, con balcón privado."),
        Room(0, "401", "Suite Ejecutiva", "Available", 350.00, "Habitación con
sala de estar, minibar y escritorio de trabajo."),
    ]

```

```

        Room(0, "503", "Suite Presidencial", "Maintenance", 600.00, "La suite
mas lujosa, con jacuzzi y comedor privado."),
        Room(0, "108", "Accesible", "Available", 150.00, "Habitacion adaptada
para personas con movilidad reducida.")
    ]
count = 0
for room in rooms_to_create:
    if room_dao.create(room):
        count += 1
        print(f"-> Habitacion '{room.getType()}' creada con ID:
{room.getId()}")
    print(f"-> {count} habitaciones creadas exitosamente.")

def populate_initial_data():
    """
    Usa los DAOs para insertar datos iniciales en la base de datos.
    Ejecuta este script una vez para llenar tus tablas.
    """
    print("Iniciando el poblado de la base de datos...")
    try:
        # Importaciones tardías para asegurar que la conexión a la BD se
        establezca primero
        from dao.employee_dao import EmployeeDAO
        from dao.ServiceDAO import ServiceDAO
        from dao.customer_dao import CustomerDAO
        from dao.room_dao import RoomDAO

        daos = {
            "employee": EmployeeDAO(),
            "service": ServiceDAO(),
            "customer": CustomerDAO(),
            "room": RoomDAO()
        }

        populate_employees(daos["employee"])
        populate_customers(daos["customer"])
        populate_services(daos["service"])
        populate_rooms(daos["room"])

        print("\nPoblado de la base de datos completado exitosamente!")

    except Exception as e:
        print(f"\nCRITICAL ERROR durante el poblado de la base de datos: {e}")
        print("Por favor, verifica la conexión a la BD, las variables de entorno
y que las tablas existan.")

```

```
if __name__ == '__main__':
    populate_initial_data()
```

## Receptionist

```
from employee import Employee

class Receptionist(Employee):
    def createReservation(self, reservation):
        super().createReservation(reservation)

    def modifyReservation(self, reservation, newCheckIn=None,
newCheckOut=None):
        super().modifyReservation(reservation, newCheckIn, newCheckOut)
```

## Reservation

```
class Reservation:
    def __init__(self, id, checkIn, checkOut, customer, room, payment=None):
        self.__id = id
        self.__checkIn = checkIn
        self.__checkOut = checkOut
        self.__customer = customer
        self.__room = room
        self.__payment = payment
        self.__services = []

    def getId(self): return self.__id
    def setId(self, id): self.__id = id

    def getCheckIn(self): return self.__checkIn
    def setCheckIn(self, checkIn): self.__checkIn = checkIn

    def getCheckOut(self): return self.__checkOut
    def setCheckOut(self, checkOut): self.__checkOut = checkOut

    def getCustomer(self): return self.__customer
    def setCustomer(self, customer): self.__customer = customer

    def getRoom(self): return self.__room
    def setRoom(self, room): self.__room = room

    def getPayment(self): return self.__payment
    def setPayment(self, payment): self.__payment = payment
```

```

def getServices(self): return self.__services

def createReservation(self):
    if self.__room.getStatus() == "Available":
        self.__room.assignCustomer(self.__customer)
        self.__customer.makeReservation(self)
        print(f"Reservation #{self.__id} created successfully for {self.__customer.getName()}")
        return True
    else:
        print(f"Unable to create reservation. Room {self.__room.getId()} not available.")
        return False

def cancelReservation(self):
    self.__room.releaseRoom()
    print(f"Reservation #{self.__id} cancelled. Room {self.__room.getId()} released.")

def modifyReservation(self, newCheckIn=None, newCheckOut=None):
    if newCheckIn:
        self.__checkIn = newCheckIn
        print(f"Reservation #{self.__id} check-in date updated to {newCheckIn}.")
    if newCheckOut:
        self.__checkOut = newCheckOut
        print(f"Reservation #{self.__id} check-out date updated to {newCheckOut}.")

def addService(self, service):
    self.__services.append(service)
    print(f"Service '{service.getType()}' added to reservation #{self.__id}.")

def showInfo(self):
    customer_name = self.__customer.getName() if hasattr(self.__customer, 'getName') else 'Unknown Customer'
    room_id = self.__room.getId() if hasattr(self.__room, 'getId') else 'Unknown Room'
    info = (
        f"--- Information of Reservation #{self.__id} ---\n"
        f"  Customer: {customer_name}\n"
        f"  Room: {room_id} ({self.__room.getType()})\n"
        f"  Check-in: {self.__checkIn} | Check-out: {self.__checkOut}\n"
        f"  Additional Services: {len(self.__services)}"
    )

```

```
)  
print(info)  
return info
```

## ReservationService

```
class ServiceReservation:  
    def __init__(self, id, date_time, customer, service):  
        self.__id = id  
        self.__date_time = date_time  
        self.__customer = customer  
        self.__service = service  
  
    def getId(self):  
        return self.__id  
  
    def getDate(self):  
        return self.__date_time  
  
    def getCustomer(self):  
        return self.__customer  
  
    def getService(self):  
        return self.__service  
  
    def setDate(self, date_time):  
        self.__date_time = date_time  
  
    def createReservation(self):  
        self.__customer.makeServiceReservation(self)  
        print(f"Service Reservation #{self.__id} created successfully for  
{self.__customer.getName()}")  
        return True  
  
    def showInfo(self):  
        return (  
            f"--- [SERVICE] Reservation #{self.__id} ---\n"  
            f"          Customer:      {self.__customer.getName()}\n"  
            f"          Email:       {self.__customer.getEmail()}\n"  
            f"          Service:     {self.__service.getType()} | Cost:  
${self.__service.getCost():.2f}\n"  
            f"          Requested Date/Time: {self.__date_time}"  
        )
```

## Room

```
class Room:
    def __init__(self, id, room_number, type, status="Available", cost=0.0, description=""):
        self.__id = id
        self.__room_number = room_number
        self.__type = type
        self.__status = status
        self.__cost = cost
        self.__description = description

    def getId(self): return self.__id
    def setId(self, id): self.__id = id

    def getRoomNumber(self): return self.__room_number
    def setRoomNumber(self, room_number): self.__room_number = room_number

    def getType(self): return self.__type
    def setType(self, type): self.__type = type

    def getStatus(self): return self.__status
    def setStatus(self, status): self.__status = status

    def getCost(self): return self.__cost
    def setCost(self, cost): self.__cost = cost

    def getDescription(self): return self.__description
    def setDescription(self, description): self.__description = description

    def assignCustomer(self, customer):
        if self.__status == "Available":
            self.__status = "Not available"
            print(f"Room {self.__id} assigned to {customer.getName()}.")
        else:
            print(f"Room {self.__id} is not available for {customer.getName()}.")  

{customer.getName()}.")

    def releaseRoom(self):
        self.__status = "Available"
        print(f"Room {self.__id} released.")

    def showInfo(self):
        print(f"Room {self.__id} - Type: {self.__type} - Status: {self.__status} - Cost: ${self.__cost} - Description: {self.__description}")
```

## Service

```
class Service:
    def __init__(self, id, type, cost, description):
        self.__id = id
        self.__type = type
        self.__cost = cost
        self.__description = description

    def getId(self): return self.__id
    def setId(self, id): self.__id = id

    def getType(self): return self.__type
    def setType(self, type): self.__type = type

    def getCost(self): return self.__cost
    def setCost(self, cost): self.__cost = cost

    def getDescription(self): return self.__description
    def setDescription(self, description): self.__description = description

    def addService(self):
        print(f"Service '{self.__type}' added. Cost: ${self.__cost}")

    def removeService(self):
        print(f"Service '{self.__type}' removed. Cost: ${self.__cost}")

    def showInfo(self):
        print(f"Service: {self.__type} - {self.__description} - ${self.__cost}")
```

## Customer\_dao

```
import mysql.connector
from db_connection import get_conn, close_conn
from customer import Customer
from typing import Optional

class CustomerDAO:
    def create(self, cust: Customer) -> Optional[int]:
        conn = None
        cursor = None
        try:
            conn = get_conn()
            cursor = conn.cursor()
```

```

        query = """
            INSERT INTO CUSTOMERS
                (first_name, second_name, last_name, second_last_name, phone,
email, state, curp, password_hash)
                VALUES (%s, %s, %s, %s, %s, %s, %s, %s)
"""
        values = (
            cust.getName(), cust.getSecondName(), cust.getLastName(),
            cust.getSecondLastName(), cust.getPhone(), cust.getEmail(),
            cust.getState(), cust.getCurp(), cust.getPassword()
        )

        cursor.execute(query, values)
        conn.commit()

        cust_id = cursor.lastrowid
        cust.setId(cust_id)

        print(f"INFO: Cliente '{cust.getName()}' insertado en la BD con ID: {cust_id}")
        return cust_id

    except mysql.connector.Error as err:
        print(f"ERROR: No se pudo crear el cliente en la BD: {err}")
        if conn:
            conn.rollback()
        return None
    finally:
        if cursor:
            cursor.close()
        if conn:
            close_conn(conn)

def get_all(self) -> list[Customer]:
    conn = None
    cursor = None
    customers = []
    try:
        conn = get_conn()
        cursor = conn.cursor()
        query = "SELECT customer_id, first_name, second_name, last_name,
second_last_name, phone, email, state, curp, password_hash FROM CUSTOMERS"
        cursor.execute(query)
        records = cursor.fetchall()
        for record in records:
            (cust_id, name, sName, lName, sLName, phone, email, state, curp,
password) = record

```

```

        customers.append(
            Customer(cust_id, name, sName, lName, sLName, phone, email,
state, curp, password)
        )
    print(f"INFO: Se cargaron {len(customers)} clientes desde la BD.")
except mysql.connector.Error as err:
    print(f"ERROR: No se pudieron obtener los clientes de la BD: {err}")
finally:
    if cursor:
        cursor.close()
    if conn:
        close_conn(conn)
return customers

```

## Employee\_dao

```

import mysql.connector
from db_connection import get_conn, close_conn
from employee import Employee
from receptionist import Receptionist
from bellboy import Bellboy
from typing import Optional, List

class EmployeeDAO:
    ROLE_MAPPING = {
        "Receptionist": Receptionist,
        "Bellboy": Bellboy,
        "Employee": Employee,
        "Manager": Employee
    }

    def create(self, emp: Employee) -> Optional[int]:
        conn = None
        cursor = None
        role = type(emp).__name__
        if role not in self.ROLE_MAPPING: # type: ignore
            role = "Employee"

        query = """
            INSERT INTO EMPLOYEES
            (first_name, second_name, last_name, second_last_name, phone,
email, curp, password_hash, status, role)
            VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s)
        """
        values = (
            emp.getFirstName(), emp.getSecondName(), emp.getLastName(),
            emp.getSecondLastName(), emp.getPhone(), emp.getEmail(),

```

```
        emp.getCurp(), emp.getPassword(), emp.getStatus(), role
    )

try:
    conn = get_conn()
    cursor = conn.cursor()
    cursor.execute(query, values)
    conn.commit()
    emp_id = cursor.lastrowid
    emp.setId(emp_id)
    return emp_id
except mysql.connector.Error as err:
    print(f"Error CREATE Employee: {err}")
    if conn:
        conn.rollback()
    return None
finally:
    if cursor:
        cursor.close()
    if conn:
        close_conn(conn)

def get_by_id(self, emp_id: int) -> Optional[Employee]:
    conn = get_conn()
    cursor = conn.cursor()

    query = """
        SELECT      employee_id,      first_name,      second_name,      last_name,
second_last_name,
                    phone, email, status, curp, password_hash, role
        FROM EMPLOYEES
        WHERE employee_id = %s
    """
    """

    try:
        cursor.execute(query, (emp_id,))
        record = cursor.fetchone()

        if record:
            (id, fName, sName, lName, sLName, phone, email, status, curp,
password, role) = record
            EmployeeClass = self.ROLE_MAPPING.get(role, Employee)
            return EmployeeClass(id, fName, sName, lName, sLName, phone,
email, status, curp, password)
        return None
    except mysql.connector.Error as err:
        print(f"Error READ Employee: {err}")
```

```

        return None
    finally:
        cursor.close()
        close_conn(conn)

    def get_all(self) -> List[Employee]:
        conn = None
        cursor = None
        employees = []
        query = """
            SELECT      employee_id,      first_name,      second_name,      last_name,
second_last_name,
                        phone, email, status, curp, password_hash, role
            FROM EMPLOYEES
        """
        try:
            conn = get_conn()
            cursor = conn.cursor()
            cursor.execute(query)
            records = cursor.fetchall()
            for record in records:
                (id, fName, sName, lName, sLName, phone, email, status, curp,
password, role) = record
                EmployeeClass = self.ROLE_MAPPING.get(role, Employee)
                employees.append(
                    EmployeeClass(id, fName, sName, lName, sLName, phone,
email, status, curp, password)
                )
            print(f"INFO: Se cargaron {len(employees)} empleados desde la BD.")
        except mysql.connector.Error as err:
            print(f"Error READ ALL Employees: {err}")
        finally:
            if cursor:
                cursor.close()
            if conn:
                close_conn(conn)
        return employees

    def update_status(self, emp_id: int, new_status: str) -> bool:
        conn = get_conn()
        cursor = conn.cursor()
        query = "UPDATE EMPLOYEES SET status=%s WHERE employee_id=%s"
        values = (new_status, emp_id)
        try:
            cursor.execute(query, values)
            conn.commit()
            return cursor.rowcount > 0
        
```

```

        except mysql.connector.Error as err:
            print(f"Error UPDATE Employee Status: {err}")
            conn.rollback()
            return False
        finally:
            cursor.close()
            close_conn(conn)

    def delete(self, emp_id: int) -> bool:
        conn = get_conn()
        cursor = conn.cursor()
        query = "DELETE FROM EMPLOYEES WHERE employee_id = %s"
        try:
            cursor.execute(query, (emp_id,))
            conn.commit()
            return cursor.rowcount > 0
        except mysql.connector.Error as err:
            print(f"Error DELETE Employee: {err}")
            conn.rollback()
            return False
        finally:
            cursor.close()
            close_conn(conn)

```

## Payment\_dao

```

import mysql.connector
from db_connection import get_conn, close_conn
from payment import Payment
from typing import Optional

class PaymentDAO:

    def create(self, payment: Payment) -> Optional[int]:
        conn = None
        cursor = None

        query = """
            INSERT INTO PAYMENTS
            (amount, payment_method, payment_date)
            VALUES (%s, %s, %s)
        """
        values = (
            payment.getAmount(),
            payment.getPaymentMethod(),
            payment.getDate()

```

```

    )

try:
    conn = get_conn()
    cursor = conn.cursor()
    cursor.execute(query, values)
    conn.commit()

    payment_id = cursor.lastrowid
    payment.setId(payment_id)

    print(f"INFO: Pago #{payment_id} por ${payment.getAmount()}")
    guardado en la BD.")
    return payment_id

except mysql.connector.Error as err:
    print(f"ERROR: No se pudo guardar el pago en la BD: {err}")
    if conn:
        conn.rollback()
    return None
finally:
    if cursor:
        cursor.close()
    if conn:
        close_conn(conn)

```

## Reservation\_dao

```

import mysql.connector
from db_connection import get_conn, close_conn
from customer import Customer
from room import Room
from reservation import Reservation
from typing import Optional, List

class ReservationDAO:

    def create(self, res: Reservation, total_cost: float) -> Optional[int]:
        conn = None
        cursor = None
        query = """
            INSERT INTO RESERVATIONS
            (customer_id, room_id, check_in_date, check_out_date, status,
            total_cost)
            VALUES (%s, %s, %s, %s, %s, %s)
        """

```

```

values = (
    res.getCustomer().getId(),
    res.getRoom().getId(),
    res.getCheckIn(),
    res.getCheckOut(),
    'Confirmed',
    total_cost
)

try:
    conn = get_conn()
    cursor = conn.cursor()
    cursor.execute(query, values)
    conn.commit()

    res_id = cursor.lastrowid
    res.setId(res_id)

    print(f"INFO: Reserva #{res_id} creada en la BD.")
    return res_id

except mysql.connector.Error as err:
    print(f"ERROR: No se pudo crear la reserva en la BD: {err}")
    if conn:
        conn.rollback()
    return None
finally:
    if cursor:
        cursor.close()
    if conn:
        close_conn(conn)

def get_all(self) -> List[Reservation]:
    conn = None
    cursor = None
    reservations = []
    query = """
        SELECT
            r.reservation_id, r.check_in_date, r.check_out_date, r.status,
            r.total_cost,
            c.customer_id, c.first_name, c.second_name, c.last_name,
            c.second_last_name, c.phone, c.email, c.state, c.curp, c.password_hash,
            rm.room_id, rm.room_number, rm.room_type, rm.status,
            rm.cost_per_night, rm.description
        FROM RESERVATIONS r
        JOIN CUSTOMERS c ON r.customer_id = c.customer_id
        JOIN ROOMS rm ON r.room_id = rm.room_id
    """

```

```

"""
try:
    conn = get_conn()
    cursor = conn.cursor()
    cursor.execute(query)
    records = cursor.fetchall()

    for record in records:
        customer = Customer(record[5], record[6], record[7], record[8],
record[9], record[10], record[11], record[12], record[13], record[14])
        room = Room(record[15], record[16], record[17], record[18],
record[19], record[20])

        reservation = Reservation(record[0], str(record[1]),
str(record[2]), customer, room)
        reservations.append(reservation)

    print(f"INFO: Se cargaron {len(reservations)} reservas desde la
BD.")

except mysql.connector.Error as err:
    print(f"ERROR: No se pudieron obtener las reservas de la BD: {err}")
finally:
    if cursor:
        cursor.close()
    if conn:
        close_conn(conn)
return reservations

def delete(self, reservation_id: int) -> bool:
    conn = None
    cursor = None
    query = "DELETE FROM RESERVATIONS WHERE reservation_id = %s"
    try:
        conn = get_conn()
        cursor = conn.cursor()
        cursor.execute(query, (reservation_id,))
        conn.commit()

        if cursor.rowcount > 0:
            print(f"INFO: Reserva #{reservation_id} eliminada de la BD.")
            return True
        return False
    except mysql.connector.Error as err:
        print(f"ERROR: No se pudo eliminar la reserva #{reservation_id}:
{err}")
        if conn:
            conn.rollback()

```

```

        return False
    finally:
        if cursor:
            cursor.close()
        if conn:
            close_conn(conn)

    def link_payment(self, reservation_id: int, payment_id: int) -> bool:
        conn = None
        cursor = None
        query = "UPDATE RESERVATIONS SET payment_id = %s WHERE reservation_id"
= %s"
        values = (payment_id, reservation_id)

        try:
            conn = get_conn()
            cursor = conn.cursor()
            cursor.execute(query, values)
            conn.commit()

            if cursor.rowcount > 0:
                print(f"INFO: Pago #{payment_id} asociado a la Reserva #{reservation_id}.")
                return True
            else:
                print(f"WARNING: No se encontró la Reserva #{reservation_id} para asociar el pago.")
                return False

        except mysql.connector.Error as err:
            print(f"ERROR: No se pudo asociar el pago a la reserva: {err}")
            if conn:
                conn.rollback()
            return False
        finally:
            if cursor:
                cursor.close()
            if conn:
                close_conn(conn)

```

## Room\_dao

```

import mysql.connector
from db_connection import get_conn, close_conn
from room import Room
from typing import List, Optional

```

```
class RoomDAO:

    def create(self, room: Room) -> Optional[int]:
        conn = None
        cursor = None
        try:
            conn = get_conn()
            cursor = conn.cursor()
            query = "INSERT INTO ROOMS (room_number, room_type, status, cost_per_night, description) VALUES (%s, %s, %s, %s, %s)"
            values = (room.getRoomNumber(), room.getType(), room.getStatus(), room.getCost(), room.getDescription())

            cursor.execute(query, values)
            conn.commit()

            room_id = cursor.lastrowid
            room.setId(room_id)
            print(f"INFO: Habitacion creada en la BD con ID: {room_id}.")
            return room.getId()
        except mysql.connector.Error as err:
            print(f"ERROR: No se pudo crear la habitacion {room.getId()}: {err}")
        if conn:
            conn.rollback()
        return None
    finally:
        if cursor:
            cursor.close()
        if conn:
            close_conn(conn)

    def get_all(self) -> List[Room]:
        conn = None
        cursor = None
        rooms = []
        try:
            conn = get_conn()
            cursor = conn.cursor()
            query = "SELECT room_id, room_number, room_type, status, cost_per_night, description FROM ROOMS"
            cursor.execute(query)
            records = cursor.fetchall()
            for record in records:
                (room_id, room_number, room_type, status, cost, description) = record
```

```

        rooms.append(Room(room_id, room_number, room_type, status,
cost, description))
    print(f"INFO: Se cargaron {len(rooms)} habitaciones desde la BD.")
except mysql.connector.Error as err:
    print(f"ERROR: No se pudieron obtener las habitaciones de la BD:
{err}")
finally:
    if cursor:
        cursor.close()
    if conn:
        close_conn(conn)
return rooms

```

## Service\_dao

```

import mysql.connector
from db_connection import get_conn, close_conn
from service import Service
from typing import Optional, List

class ServiceDAO:

    def create(self, svc: Service) -> Optional[int]:
        conn = None
        cursor = None

        query = """
            INSERT INTO SERVICES
            (name, cost, description)
            VALUES (%s, %s, %s)
        """
        values = (
            svc.getType(), svc.getCost(), svc.getDescription()
        )

        conn = None
        try:
            conn = get_conn()
            with conn.cursor() as cursor:
                cursor.execute(query, values)
                conn.commit()
                svc_id = cursor.lastrowid
                svc.setId(svc_id)
                return svc_id
        except mysql.connector.Error as err:

```

```

        print(f"Error CREATE Service: {err}")
        if conn:
            conn.rollback()
        return None
    finally:
        if conn:
            close_conn(conn)

    def get_by_id(self, service_id: int) -> Optional[Service]:
        conn = get_conn()
        cursor = conn.cursor()

        query = "SELECT service_id, name, cost, description FROM SERVICES WHERE
service_id = %s"

        try:
            cursor.execute(query, (service_id,))
            record = cursor.fetchone()

            if record:
                (id, type_name, cost, description) = record
                return Service(id, type_name, cost, description)
            return None
        except mysql.connector.Error as err:
            print(f"Error READ Service: {err}")
            return None
    finally:
        cursor.close()
        close_conn(conn)

    def get_all(self) -> List[Service]:
        conn = None
        cursor = None
        services = []
        query = "SELECT service_id, name, cost, description FROM SERVICES"
        try:
            conn = get_conn()
            cursor = conn.cursor()
            cursor.execute(query)
            records = cursor.fetchall()
            for id, type_name, cost, description in records:
                services.append(Service(id, type_name, cost, description))
            print(f"INFO: Se cargaron {len(services)} servicios desde la BD.")
        except mysql.connector.Error as err:
            print(f"Error READ ALL Services: {err}")
        finally:
            if cursor:

```

```

        cursor.close()
    if conn:
        close_conn(conn)
    return services

def delete(self, service_id: int) -> bool:
    conn = get_conn()
    cursor = conn.cursor()
    query = "DELETE FROM SERVICES WHERE service_id = %s"

    try:
        cursor.execute(query, (service_id,))
        conn.commit()
        return cursor.rowcount > 0
    except mysql.connector.Error as err:
        print(f"Error DELETE Service: {err}")
        conn.rollback()
        return False
    finally:
        cursor.close()
        close_conn(conn)

```

## .env

```

# .env
DB_HOST=localhost
DB_PORT=3306
DB_NAME=hotel_le_villa
DB_USER=root
DB_PASSWORD=admin
POOL_SIZE=5

```

## Explicación detallada de los códigos

El diseño del sistema está basado en una sólida **Separación de Responsabilidades**, dividiendo el código en Entidades de Dominio, Objetos de Acceso a Datos (DAO) y una capa de Infraestructura, utilizando el **Encapsulamiento** (atributos privados con \_) como mecanismo principal de integridad.

### I. Entidades de Dominio (Domain Entities)

Estas clases modelan los conceptos clave del negocio y contienen la lógica de colaboración entre objetos.

## **1. Clase Customer**

- **Propósito:** Representar al principal actor del sistema, el huésped.
- **Estructura Interna:** Almacena atributos personales (nombre, CURP, correo, etc.) y dos listas cruciales: `_reservations` y `_service_reservations`. Esto convierte al cliente en el **objeto raíz de agregación** para todas sus interacciones de reserva.
- **Colaboración:**
  - El método `makeReservation(reservation)` ejemplifica la **cohesión**, ya que es responsable de mantener la lista de reservas del cliente sincronizada con la reserva recién creada.
  - Los métodos de cancelación (`cancelReservation`, `cancelServiceReservation`) utilizan la **identificación de objeto** (`reservation_id`) para gestionar el estado interno de la lista.

## **2. Clase Employee**

- **Propósito:** Servir como la **clase base abstracta** para todo el personal, definiendo el contrato común para la manipulación de reservas.
- **Herencia y Delegación:** Sus métodos de acción (`createReservation`, `cancelReservation`, `modifyReservation`) demuestran el patrón **Delegación**. El empleado no contiene la lógica de cómo se crea o se cancela una reserva; en su lugar, delega la ejecución al objeto `Reservation` que se le pasa como argumento (`reservation.createReservation()`).
- **Atributos de Rol:** Incluye atributos administrativos (`_status`, `_password`) que son fundamentales para la autenticación y el mapeo de roles en el DAO.

## **3. Clase Receptionist**

- **Propósito:** Especializar el rol de `Employee` para las tareas de *front-desk*.
- **Aplicación de POO:** Es una subclase directa de `Employee`. Los métodos sobrescritos (`createReservation`, `modifyReservation`) que llaman a `super()` están listos para implementar una lógica **polimórfica** específica para el recepcionista (p. ej., generar una factura preliminar, registrar un evento de *check-in/out*).

## **4. Clase Reservation**

- **Propósito:** Modelar la asignación de un recurso (habitación) a un cliente en un período de tiempo.

- **Relaciones Clave (Composición):** Mantiene referencias directas a las instancias de `_customer`, `_room` y, opcionalmente, `_payment`. La existencia de la reserva depende de estos objetos.
- **Transición de Estado:** El método `createReservation()` es un ejemplo de **interacción de objetos** que maneja la **transición de estado** de recursos: verifica el estado de la `_room`, y si está disponible, llama a `_room.assignCustomer()` y `_customer.makeReservation()`.
- **Gestión de Recursos:** `cancelReservation()` utiliza la función inversa, llamando a `_room.releaseRoom()`, garantizando la consistencia del inventario de habitaciones.

## 5. Clase Payment

- **Propósito:** Abstraer los detalles de una transacción financiera.
- **Inmutabilidad Parcial:** El atributo `_date` se establece en el constructor utilizando `date.today()`, lo que refleja que el momento del pago es un hecho del pasado que no debe modificarse.
- **Comportamiento:** Los métodos `processPayment()` y `cancelPayment()` simulan la interacción con un sistema externo (una pasarela de pago), ocultando los detalles de la integración real.

## 6. Clase ServiceReservation

- **Propósito:** Clase de enlace o asociación que modela la solicitud de un servicio adicional por un cliente específico.
- **Separación de Conceptos:** Separa la reserva de habitación (`Reservation`) de la reserva de servicio (`ServiceReservation`), lo que permite una **escalabilidad** independiente de ambos subsistemas.

## II. Objetos de Acceso a Datos (DAO Layer)

Esta capa gestiona la persistencia, desacoplando la lógica de negocio del *middleware* de la base de datos.

## 7. Clases \*DAO (e.g., `CustomerDAO`, `ReservationDAO`)

- **Patrón de Diseño:** Implementan el patrón **DAO** estrictamente. Su contrato de métodos define las operaciones CRUD (Create, Read, Update, Delete).
- **Gestión Transaccional:** Cada método de escritura (create, update, delete) envuelve su operación en un bloque `try-except-finally` para asegurar el manejo correcto de la conexión (vía `close_conn`) y la **Integridad**

**Transaccional** (`conn.commit()` o `conn.rollback()` en caso de excepción `mysql.connector.Error`).

- **Mapeo Objeto-Relacional (ORM Manual):**
  - **Creación:** El método `create()` utiliza `cursor.lastrowid` para obtener el ID generado por la base de datos y luego llama a `entity.setId()`, completando el mapeo del ID de la BD al objeto en memoria.
  - **Lectura:** Los métodos `get_all()` y `get_by_id()` construyen dinámicamente nuevas instancias de las entidades (p. ej., `Customer(...)`) a partir de los registros (`records`) obtenidos del cursor, realizando el mapeo inverso.
- **EmployeeDAO (Polimorfismo en Lectura):** Su método `get_all()` utiliza el mapeo `ROLE_MAPPING` para instanciar subclases específicas (`Receptionist`, `Bellboy`) en lugar de la clase base `Employee`, logrando el **Polimorfismo de Instanciación** basado en datos.
- **ReservationDAO.link\_payment():** Este método especializado gestiona una **asociación bidireccional** entre la reserva y el pago a nivel de persistencia, actualizando el `payment_id` en la tabla `RESERVATIONS`.

### III. Capa de Infraestructura

Esta capa gestiona los recursos de bajo nivel esenciales para la operación del sistema.

#### 8. Clase DBConnection (en `db_connection.py`)

- **Manejo de Recursos:** Utiliza la librería `mysql.connector.pooling` para implementar el **Patrón Pool de Conexiones**. Esto minimiza la latencia de apertura/cierre de conexiones y optimiza el consumo de recursos del servidor de base de datos.
- **Patrón Singleton:** La instancia `_db_connection_instance` se inicializa a nivel de módulo, asegurando que solo exista un objeto `DBConnection` en la aplicación para gestionar el *pool*. Esto centraliza la administración de la conexión.
- **Inyección de Configuración:** Los parámetros de conexión (`HOST`, `USER`, `PASSWORD`, `POOL_SIZE`) son importados desde `mysql_env.py`, manteniendo la **externalización de la configuración** fuera del código binario.
- **Interfaz del Módulo:** Las funciones `get_conn()` y `close_conn()` actúan como una interfaz de proveedor de conexiones fácil de consumir por las clases DAO, manteniendo la abstracción de la lógica del *pool*.

## I. Configuración del Entorno MySQL

Esta sección detalla los pasos para asegurar que el entorno de la base de datos MySQL esté listo y correctamente configurado para la aplicación.

### 1. Instalación y Dependencias

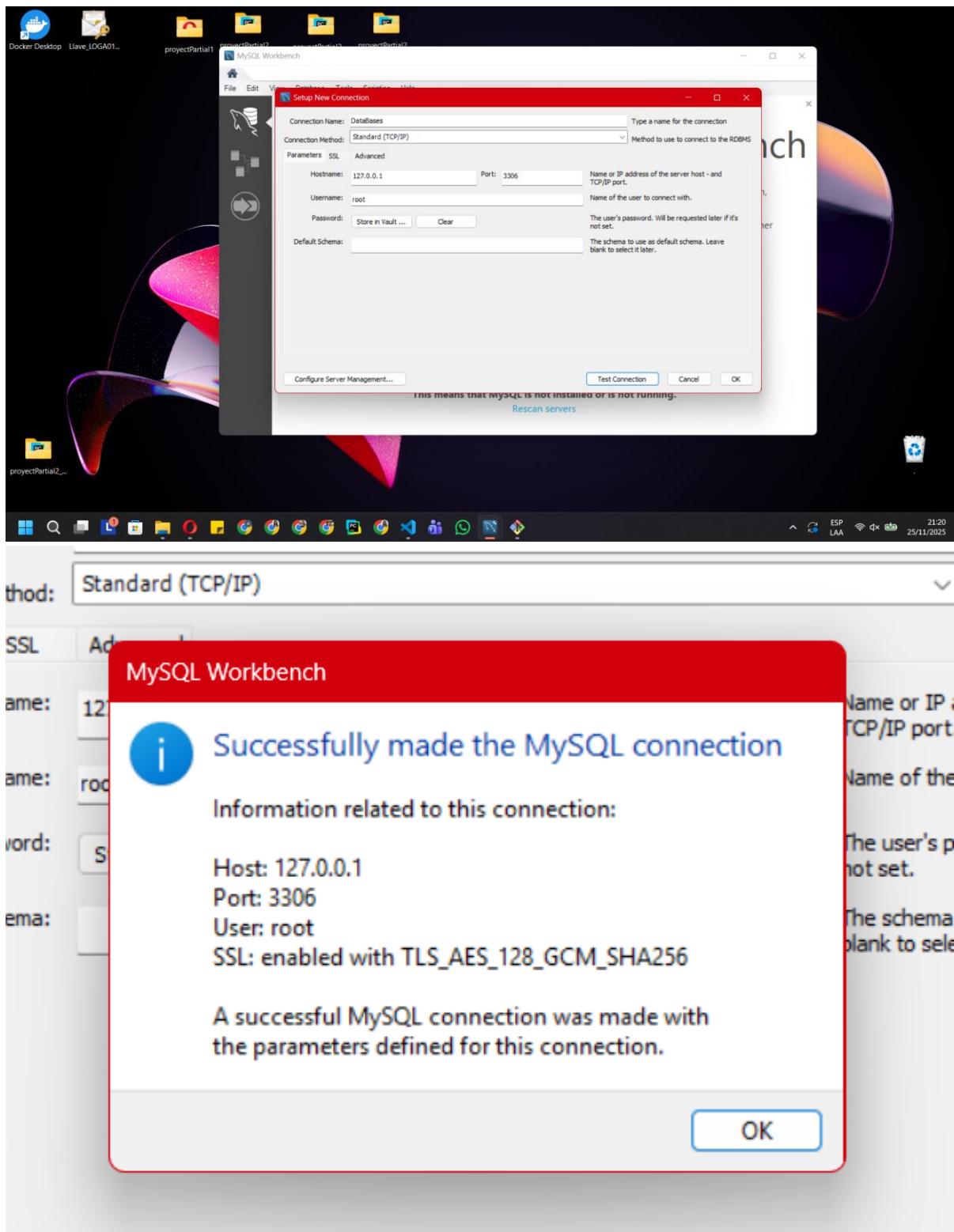
Se requiere la instalación del motor de base de datos y la librería de Python para la conectividad.

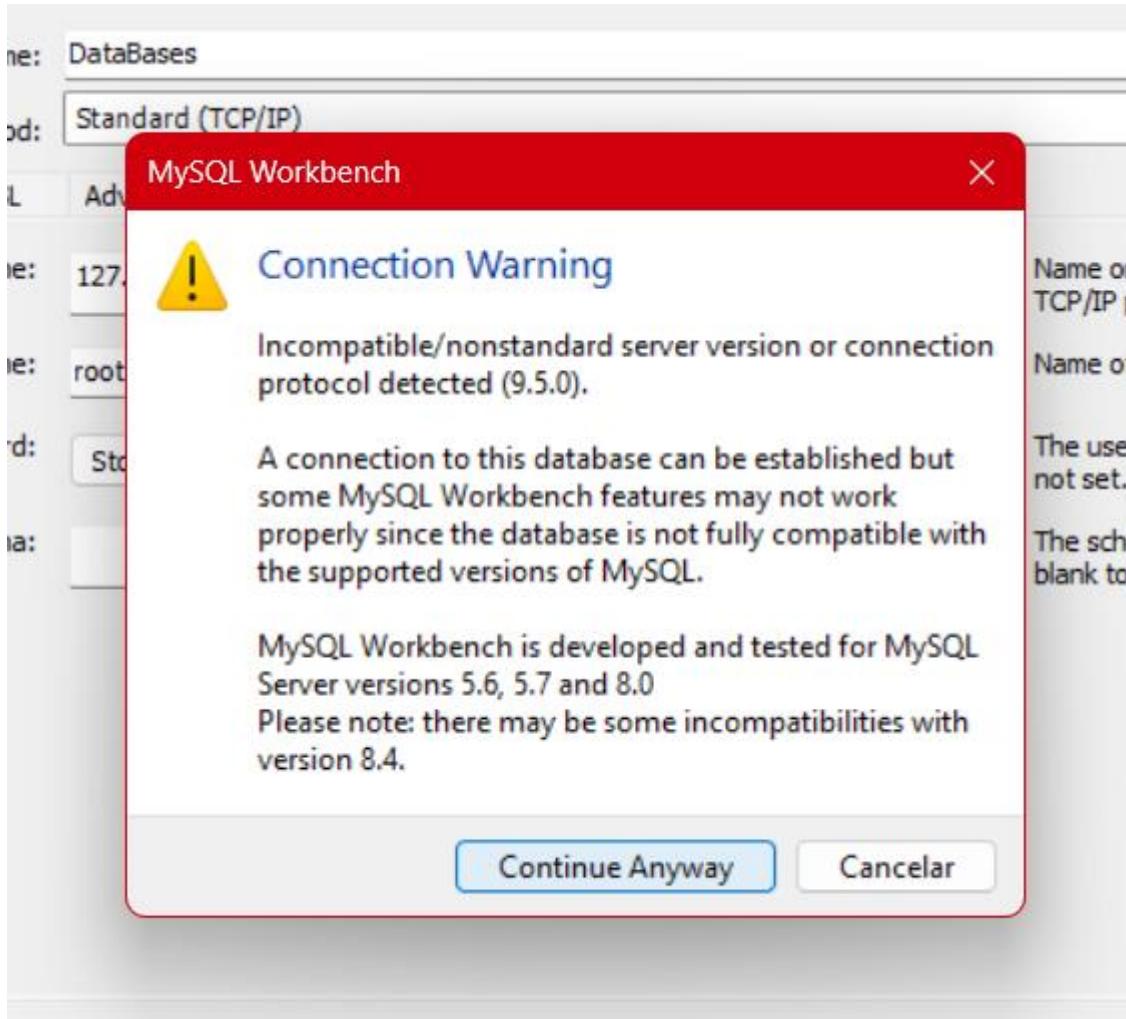
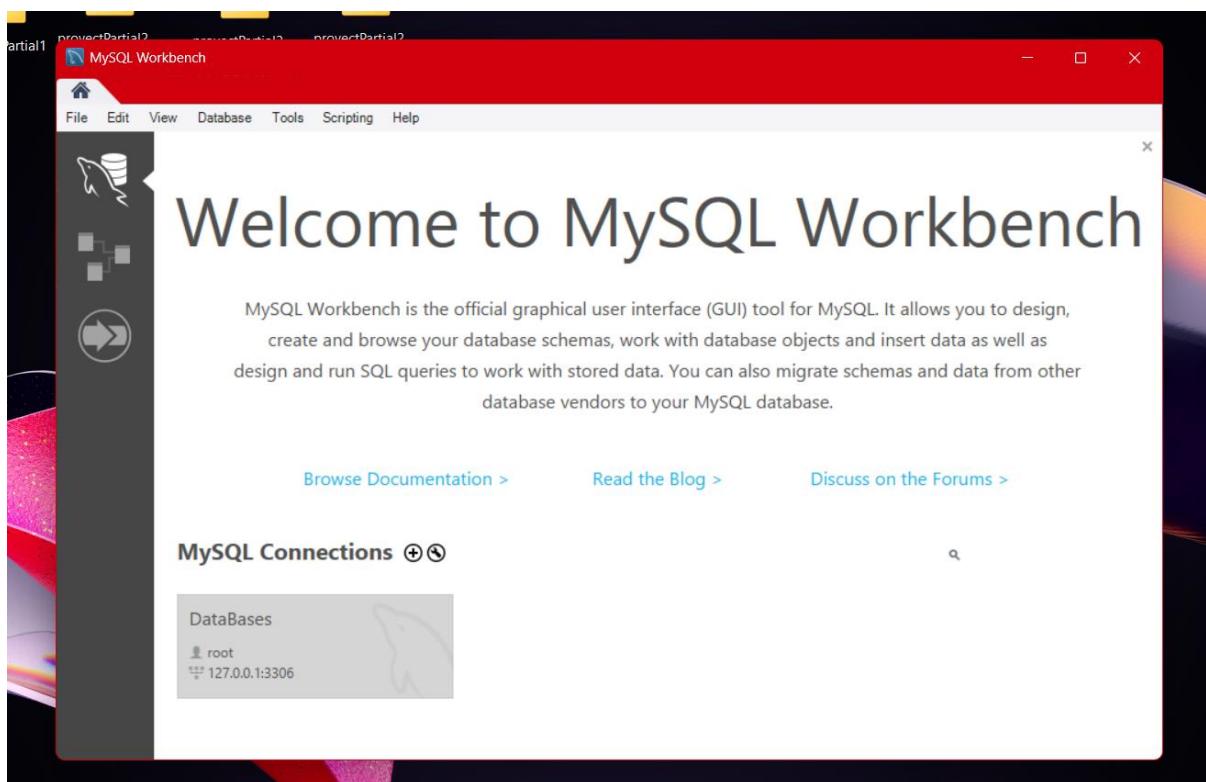
Elemento	Requisito/Comando	Notas
<b>Servidor MySQL</b>	Instalación de MySQL Server (v8.0+)	Se recomienda la instalación mediante el gestor de paquetes de tu sistema operativo (e.g., <code>sudo apt install mysql-server</code> en Debian/Ubuntu).
<b>Librería Python</b>	<code>mysql-connector-python</code>	Instalado con <code>pip install mysql-connector-python</code> . <b>Esta librería incluye el módulo pooling</b> esencial para la clase <code>DBConnection</code> .

### 2. Creación de la Base de Datos y Usuario

Los siguientes comandos SQL se ejecutan en la consola de MySQL para establecer el esquema y las credenciales de la aplicación.

Detalle	Comando SQL	Descripción
<b>Creación de la BD</b>	<code>CREATE DATABASE IF NOT EXISTS `hotel_le_villa`;</code>	Crea el esquema de la base de datos principal de la aplicación.





MySQL Workbench

DataBases - Warning - not su...

File Edit View Query Database Server Tools Scripting Help

Navigator

SCHEMAS

- hotel\_le\_villa
  - Tables
  - Views
  - Stored Procedures
  - Functions
- sys

Administration Schemas

No object selected

Query 1

```

1 USE hotel_le_villa;
2 CREATE TABLE ROOMS (
3   room_id INT AUTO_INCREMENT PRIMARY KEY,
4   room_number VARCHAR(10) NOT NULL UNIQUE, -- Using a separate number field
5   room_type VARCHAR(50) NOT NULL, -- e.g., 'Single', 'Double', 'Suite' (from getType)
6   description TEXT, -- (from getDescription)
7   cost_per_night DECIMAL(10, 2) NOT NULL, -- (from getCost)
8   status VARCHAR(20) NOT NULL DEFAULT 'Available', -- (from getStatus: 'Available', 'Not available')
9   CHECK (status IN ('Available', 'Not available', 'Maintenance'))
10 );

```

SQLAdditions

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

Output

Action Output

#	Time	Action	Message	Duration /Fetch
1	21:21:43	USE hotel_le_villa	Error Code: 1049. Unknown database 'hotel_le_villa'	0.00 sec
2	21:21:56	CREATE DATABASE IF NOT EXISTS hotel_le_villa	1 row(s) affected	0.047 sec
3	21:21:56	USE hotel_le_villa	0 row(s) affected	0.00 sec
4	21:22:49	CREATE TABLE Clientes ( id_cliente INT AUTO_INCREMENT PRIMARY KEY, nombre VARCHAR(50) NOT N... )	0 row(s) affected	0.094 sec
5	21:33:49	DROP TABLE Clientes	0 row(s) affected	0.171 sec

Object Info Session

MySQL Workbench

DataBases - Warning - not su...

File Edit View Query Database Server Tools Scripting Help

Navigator

SCHEMAS

- hotel\_le\_villa
  - Tables
  - Views
  - Stored Procedures
  - Functions
- sys

Administration Schemas

No object selected

Query 1

```

1 CREATE DATABASE IF NOT EXISTS hotel_le_villa;

```

SQLAdditions

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

Output

Action Output

#	Time	Action	Message	Duration /Fetch
1	21:21:43	USE hotel_le_villa	Error Code: 1049. Unknown database 'hotel_le_villa'	0.00 sec
2	21:21:56	CREATE DATABASE IF NOT EXISTS hotel_le_villa	1 row(s) affected	0.047 sec
3	21:21:56	USE hotel_le_villa	0 row(s) affected	0.00 sec
4	21:22:49	CREATE TABLE Clientes ( id_cliente INT AUTO_INCREMENT PRIMARY KEY, nombre VARCHAR(50) NOT N... )	0 row(s) affected	0.094 sec
5	21:33:49	DROP TABLE Clientes	0 row(s) affected	0.171 sec

Object Info Session

**MySQL Workbench**

DataBases - Warning - not su...

File Edit View Query Database Server Tools Scripting Help

Navigator

SCHEMAS

- hotel\_le\_villa
  - Tables
    - customers
    - employees
    - payments
    - reservations
    - rooms
    - services
  - Views
  - Stored Procedures
  - Functions
- sys

Administration Schemas

No object selected

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content: |

Query 1 | x

1  
2 • SELECT \* FROM services;

3

service_id	name	cost	description
1	Room Service 24h	15.00	Entrega de alimentos y bebidas a la habitación ...
2	Lavandería Express	30.00	Lavado, secado y planchado con entrega en 4 ...
3	Acceso a Spa	50.00	Acceso a sauna, baño turco y piscina climatizad...
4	Masaje Terapéutico	85.00	Sesión privada de masaje de cuerpo completo (...
5	Cama Adicional	25.00	Instalación de una cama supletoria en la habitaci...
6	Parqueo con Valet	10.00	Servicio de aparcacoches y recogida del vehículo.
7	Room Service 24h	15.00	Entrega de alimentos y bebidas a la habitación ...
8	Lavandería Express	30.00	Lavado, secado y planchado con entrega en 4 ...
9	Acceso a Spa	50.00	Acceso a sauna, baño turco y piscina climatizad...
10	Masaje Terapéutico	85.00	Sesión privada de masaje de cuerpo completo (...

customers 15 employees 16 employees 19 customers 20 services 21 services 42 x

Action Output

Time	Action	Message	Duration / Fetch
66 01:13:42	DROP TABLE `hotel_le_villa`.`service_reservations`	0 row(s) affected	0.250 sec
67 01:14:09	SELECT * FROM employees LIMIT 0, 50	4 row(s) returned	0.031 sec / 0.000 sec
68 01:14:20	SELECT * FROM payments LIMIT 0, 50	5 row(s) returned	0.000 sec / 0.000 sec
69 01:14:31	SELECT * FROM reservations LIMIT 0, 50	2 row(s) returned	0.031 sec / 0.000 sec
70 01:14:43	SELECT * FROM rooms LIMIT 0, 50	6 row(s) returned	0.016 sec / 0.000 sec
71 01:14:53	SELECT * FROM services LIMIT 0, 50	24 row(s) returned	0.015 sec / 0.000 sec

Object Info Session

Output

ESP LAA 1:14 26/11/2025

**MySQL Workbench**

DataBases - Warning - not su...

File Edit View Query Database Server Tools Scripting Help

Navigator

SCHEMAS

- hotel\_le\_villa
  - Tables
    - customers
    - employees
    - payments
    - reservations
    - rooms
    - services
  - Views
  - Stored Procedures
  - Functions
- sys

Administration Schemas

No object selected

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content: |

Query 1 | x

1  
2 • SELECT \* FROM rooms;

3

room_id	room_number	room_type	description	cost_per_night	status
1	101	Individual	Una cama Queen size, ideal para un viajero.	120.00	Available
2	205	Doble	Dos camas Queen size, para hasta 4 huéspedes.	180.00	Available
3	310	King Size	Una cama King size, con balcón privado.	220.00	Not available
4	401	Suite Ejecutiva	Habitación con sala de estar, minibar y escritori...	350.00	Available
5	503	Suite Presidential	La suite más lujosa, con jacuzzi y comedor priva...	600.00	Maintenance
6	108	Accessible	Habitación adaptada para personas con movilid...	150.00	Available

customers 15 employees 16 employees 19 customers 20 rooms 41 services 21 x

Action Output

Time	Action	Message	Duration / Fetch
65 01:12:22	SELECT * FROM customers LIMIT 0, 50	4 row(s) returned	0.070 sec / 0.000 sec
66 01:13:42	DROP TABLE `hotel_le_villa`.`service_reservations`	0 row(s) affected	0.250 sec
67 01:14:09	SELECT * FROM employees LIMIT 0, 50	4 row(s) returned	0.031 sec / 0.000 sec
68 01:14:20	SELECT * FROM payments LIMIT 0, 50	5 row(s) returned	0.000 sec / 0.000 sec
69 01:14:31	SELECT * FROM reservations LIMIT 0, 50	2 row(s) returned	0.031 sec / 0.000 sec
70 01:14:43	SELECT * FROM rooms LIMIT 0, 50	6 row(s) returned	0.016 sec / 0.000 sec

Object Info Session

Output

ESP LAA 1:14 26/11/2025

**MySQL Workbench**

DataBases - Warning - not su...

File Edit View Query Database Server Tools Scripting Help

Navigator

SCHEMAS

- hotel\_le\_villa
  - Tables
    - customers
    - employees
    - payments
    - reservations
    - rooms
    - services
  - Views
  - Stored Procedures
  - Functions
- sys

Administration Schemas Information

No object selected

Query 1 x

1  
2 • SELECT \* FROM reservations;

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content: |

reservation_id	customer_id	room_id	payment_id	check_in_date	check_out_date	status	total_cost
1	2	1	3	2025-11-26	2025-12-31	Confirmed	4230.00
2	1	4	5	2025-11-26	2025-12-31	Confirmed	12300.00

Result Grid Form Editor Field Types

customers 15 employees 16 employees 19 customers 20 services 21 reservations 40 x

Action Output

Time	Action	Message	Duration / Fetch
64 01:00:10	SELECT * FROM employees LIMIT 0, 50	4 row(s) returned	0.016 sec / 0.000 sec
65 01:12:22	SELECT * FROM customers LIMIT 0, 50	4 row(s) returned	0.076 sec / 0.000 sec
66 01:13:42	DROP TABLE hotel_le_villa .`service_reservations`	0 row(s) affected	0.250 sec
67 01:14:09	SELECT * FROM employees LIMIT 0, 50	4 row(s) returned	0.031 sec / 0.000 sec
68 01:14:20	SELECT * FROM payments LIMIT 0, 50	5 row(s) returned	0.000 sec / 0.000 sec
69 01:14:31	SELECT * FROM reservations LIMIT 0, 50	2 row(s) returned	0.031 sec / 0.000 sec

Object Info Session

MySQL Workbench

DataBases - Warning - not su...

File Edit View Query Database Server Tools Scripting Help

Navigator

SCHEMAS

- hotel\_le\_villa
  - Tables
    - customers
    - employees
    - payments
    - reservations
    - rooms
    - services
  - Views
  - Stored Procedures
  - Functions
- sys

Administration Schemas Information

No object selected

Query 1 x

1  
2 • SELECT \* FROM payments;

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content: |

payment_id	amount	payment_method	payment_date
1	170.00	Efectivo	2025-11-26 00:00:00
2	15.00	Efectivo	2025-11-26 00:00:00
3	4230.00	Efectivo	2025-11-26 00:00:00
4	15.00	Tarjeta de Crédito	2025-11-26 00:00:00
5	12300.00	Efectivo	2025-11-26 00:00:00

Result Grid Form Editor Field Types

customers 15 employees 16 employees 19 customers 20 services 21 payments 39 x

Action Output

Time	Action	Message	Duration / Fetch
63 00:59:20	SELECT * FROM customers LIMIT 0, 50	4 row(s) returned	0.015 sec / 0.000 sec
64 01:00:10	SELECT * FROM employees LIMIT 0, 50	4 row(s) returned	0.016 sec / 0.000 sec
65 01:12:22	SELECT * FROM customers LIMIT 0, 50	4 row(s) returned	0.078 sec / 0.000 sec
66 01:13:42	DROP TABLE hotel_le_villa .`service_reservations`	0 row(s) affected	0.250 sec
67 01:14:09	SELECT * FROM employees LIMIT 0, 50	4 row(s) returned	0.031 sec / 0.000 sec
68 01:14:20	SELECT * FROM payments LIMIT 0, 50	5 row(s) returned	0.000 sec / 0.000 sec

Object Info Session

**MySQL Workbench**

**DataBases - Warning - not su...**

**File Edit View Query Database Server Tools Scripting Help**

**Navigator**

**SCHEMAS**

- Filter objects
- hotel\_le\_villa
  - Tables
    - customers
    - employees
    - payments
    - reservations
    - rooms
    - services
  - Views
  - Stored Procedures
  - Functions
- sys

**Administration Schemas**

**Information**

No object selected

**Query 1**

```
1
2 • SELECT * FROM employees;
3
```

**Result Grid**

employee_id	first_name	second_name	last_name	second_last_name	phone	email	curp	password_hash	role
1	Samantha		Java		443598972	sam@gmail.com	SAJAB210409ORNE	admin	Receptionist
2	Brigitte	Herrera	Rodriguez		442111111	brigitte@mail.com	HE88001122QRD	brigitte123	Receptionist
3	Carlos	Gomez	Solis		442222222	carlos@mail.com	GOSCO01122QRD	carlos123	Bellboy
10	Arturo	Mondragon	Tapia		456792165679	artmon@atap@gmail.com	ARMGT241815A4QHnB	admin	Employee

**Action Output**

Time	Action	Message	Duration / Fetch
62 00:58:56	SELECT * FROM reservations LIMIT 0, 50	1 row(s) returned	0.000 sec / 0.000 sec
63 00:59:20	SELECT * FROM customers LIMIT 0, 50	4 row(s) returned	0.015 sec / 0.000 sec
64 01:00:10	SELECT * FROM employees LIMIT 0, 50	4 row(s) returned	0.016 sec / 0.000 sec
65 01:12:22	SELECT * FROM customers LIMIT 0, 50	4 row(s) returned	0.076 sec / 0.000 sec
66 01:13:42	DROP TABLE hotel_le_villa.`service_reservations`	0 row(s) affected	0.250 sec
67 01:14:09	SELECT * FROM employees LIMIT 0, 50	4 row(s) returned	0.031 sec / 0.000 sec

**Object Info Session**

**MySQL Workbench**

**DataBases - Warning - not su...**

**File Edit View Query Database Server Tools Scripting Help**

**Navigator**

**SCHEMAS**

- Filter objects
- hotel\_le\_villa
  - Tables
    - customers
    - employees
    - payments
    - reservations
    - rooms
    - services
  - Views
  - Stored Procedures
  - Functions
- sys

**Administration Schemas**

**Information**

No object selected

**Query 1**

```
1
2 • SELECT * FROM customers;
3
```

**Result Grid**

customer_id	first_name	second_name	last_name	second_last_name	phone	email	state	curp	password_hash
1	Margot		Python		44253689756	margopy@gmail.com	Argentina	MARGP47248920	admin
2	Valentina		Lopez		44353569865	valelop@gmail.com	Mexico	VALOP479212SHUQ	admin
3	Andrea	Sarah	Lopez	Guerrero	44200000000	andrea@mail.com	Querétaro	LOGA001122QRD	andrea123
4	Juan		Perez	Vera	44233133333	juan@mail.com	Querétaro	PEV001122QRD	juan123

**Action Output**

Time	Action	Message	Duration / Fetch
61 00:58:39	SELECT * FROM payments LIMIT 0, 50	4 row(s) returned	0.070 sec / 0.000 sec
62 00:58:56	SELECT * FROM reservations LIMIT 0, 50	1 row(s) returned	0.000 sec / 0.000 sec
63 00:59:20	SELECT * FROM customers LIMIT 0, 50	4 row(s) returned	0.015 sec / 0.000 sec
64 01:00:10	SELECT * FROM employees LIMIT 0, 50	4 row(s) returned	0.016 sec / 0.000 sec
65 01:12:22	SELECT * FROM customers LIMIT 0, 50	4 row(s) returned	0.076 sec / 0.000 sec
66 01:13:42	DROP TABLE hotel_le_villa.`service_reservations`	0 row(s) affected	0.250 sec

**Object Info Session**

**MySQL Workbench**

DataBases - Warning - not su...

File Edit View Query Database Server Tools Scripting Help

Navigator

SCHEMAS

hotel\_le\_villa

- Tables
  - customers
  - employees
  - Columns
  - Indexes
  - Foreign Keys
  - Triggers
  - payments

Administration Schemas

Information

Table: customers

Columns:

customer_id	int AI PK
first_name	varchar
last_name	varchar
second_last_name	varchar
phone	varchar
email	varchar
state	varchar
curp	varchar
password_hash	varchar

Query 1

```
1 • SELECT * FROM customers;
```

Result Grid | Filter Rows | Edit | Export/Import | Wrap Cell Content |

customer_id	first_name	second_name	last_name	second_last_name	phone	email	state	curp	password_hash
1	John	Doe	Jane		123-4567890	john.doe@example.com	CA	ABC123	hashed_password

SQLAdditions | Jump to

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

Result Grid | Form Editor | Field Types | Context Help | Snippets

Action Output

#	Time	Action	Message	Duration / Fetch
17	22:06:20	USE TABLE employees	Error Code: 1064. You have an error in your SQL syntax; check the manual that corresponds to your MySQL se...	0.000 sec
18	22:08:40	SELECT * FROM employees LIMIT 0, 50	0 row(s) returned	0.110 sec / 0.000 sec
19	22:09:04	SELECT * FROM reservations LIMIT 0, 50	0 row(s) returned	0.016 sec / 0.000 sec
20	22:42:12	SELECT * FROM customers LIMIT 0, 50	0 row(s) returned	0.109 sec / 0.000 sec
21	22:44:07	SELECT * FROM reservation_services LIMIT 0, 50	0 row(s) returned	0.016 sec / 0.000 sec
22	22:49:30	SELECT * FROM customers LIMIT 0, 50	0 row(s) returned	0.031 sec / 0.000 sec

Object Info Session

22:49 25/11/2025 ESP LAA

**MySQL Workbench**

DataBases - Warning - not su...

File Edit View Query Database Server Tools Scripting Help

Navigator

SCHEMAS

hotel\_le\_villa

- Tables
  - customers
  - employees
  - payments
  - reservations
  - rooms
  - service\_reservations
  - services

Administration Schemas

Information

No object selected

Query 1

```
1 USE hotel_le_villa;
2 • CREATE TABLE RESERVATION_SERVICES (
  reservation_id INT NOT NULL,
  service_id INT NOT NULL,
  quantity INT NOT NULL DEFAULT 1,
  PRIMARY KEY (reservation_id, service_id),
  FOREIGN KEY (reservation_id) REFERENCES RESERVATIONS(reservation_id) ON DELETE CASCADE,
  FOREIGN KEY (service_id) REFERENCES SERVICES(service_id) ON DELETE RESTRICT
10 );
```

Result Grid | Filter Rows | Edit | Export/Import | Wrap Cell Content |

SQLAdditions | Jump to

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

Result Grid | Form Editor | Field Types | Context Help | Snippets

Action Output

#	Time	Action	Message	Duration / Fetch
10	21:50:09	CREATE TABLE SERVICES ( service_id INT AUTO_INCREMENT PRIMARY KEY, name VARCHAR(100) N...	0 row(s) affected	0.047 sec
11	21:50:47	CREATE TABLE PAYMENTS ( payment_id INT AUTO_INCREMENT PRIMARY KEY, - from getid) amount ...	0 row(s) affected	0.047 sec
12	21:51:13	CREATE TABLE RESERVATIONS ( reservation_id INT AUTO_INCREMENT PRIMARY KEY, - from getid) c...	0 row(s) affected	0.109 sec
13	21:51:29	CREATE TABLE SERVICE_RESERVATIONS ( service_reservation_id INT AUTO_INCREMENT PRIMARY KE...	0 row(s) affected	0.079 sec
14	21:51:47	CREATE TABLE RESERVATION_SERVICES ( reservation_id INT NOT NULL, service_id INT NOT NULL, ...	0 row(s) affected	0.078 sec

Object Info Session

21:51 25/11/2025 ESP LAA

**MySQL Workbench**

**Query 1**

```

USE hotel_le_villa;
CREATE TABLE SERVICE_RESERVATIONS (
    service_reservation_id INT AUTO_INCREMENT PRIMARY KEY, -- (from getId)
    customer_id INT NOT NULL, -- (from getCustomer)
    service_id INT NOT NULL, -- (from getService)
    reservation_id INT, -- LLinks to the main room reservation (optional, if service is booked without a main reservation)
    date_time DATETIME NOT NULL, -- (from getDate)
    status VARCHAR(20) NOT NULL DEFAULT 'Requested',
);

FOREIGN KEY (customer_id) REFERENCES CUSTOMERS(customer_id) ON DELETE CASCADE,
FOREIGN KEY (service_id) REFERENCES SERVICES(service_id) ON DELETE RESTRICT,
FOREIGN KEY (reservation_id) REFERENCES RESERVATIONS(reservation_id) ON DELETE SET NULL
);

```

**Output**

#	Time	Action	Message	Duration / Fetch
9	21:49:49	CREATE TABLE CUSTOMERS ( customer_id INT AUTO_INCREMENT PRIMARY KEY, -- from getId) first_n...	0 row(s) affected	0.054 sec
10	21:50:09	CREATE TABLE SERVICES ( service_id INT AUTO_INCREMENT PRIMARY KEY, name VARCHAR(100) N...	0 row(s) affected	0.047 sec
11	21:50:47	CREATE TABLE PAYMENTS ( payment_id INT AUTO_INCREMENT PRIMARY KEY, -- from getId) amount ...	0 row(s) affected	0.047 sec
12	21:51:13	CREATE TABLE RESERVATIONS ( reservation_id INT AUTO_INCREMENT PRIMARY KEY, -- from getId) c...	0 row(s) affected	0.109 sec
13	21:51:29	CREATE TABLE SERVICE_RESERVATIONS ( service_reservation_id INT AUTO_INCREMENT PRIMARY KE...	0 row(s) affected	0.079 sec

**Object Info** **Session**

**MySQL Workbench**

**Query 1**

```

USE hotel_le_villa;
CREATE TABLE RESERVATIONS (
    reservation_id INT AUTO_INCREMENT PRIMARY KEY, -- (from getId)
    customer_id INT NOT NULL, -- (from getCustomer)
    room_id INT NOT NULL, -- (from getRoom)
    payment_id INT, -- Optional link (from getPayment)
    check_in_date DATE NOT NULL, -- (from getCheckIn)
    check_out_date DATE NOT NULL, -- (from getCheckout)
    status VARCHAR(20) NOT NULL DEFAULT 'Pending', -- 'Pending', 'Confirmed', 'Checked_In', 'Checked_Out', 'Cancelled'
    total_cost DECIMAL(10, 2) NOT NULL,
);

FOREIGN KEY (customer_id) REFERENCES CUSTOMERS(customer_id) ON DELETE RESTRICT,
FOREIGN KEY (room_id) REFERENCES ROOMS(room_id) ON DELETE RESTRICT,
-- Payment is linked here. We use ON DELETE SET NULL to keep payment record if reservation is deleted (for auditing)
FOREIGN KEY (payment_id) REFERENCES PAYMENTS(payment_id) ON DELETE SET NULL,
CHECK (check_out_date > check_in_date)
);

```

**Output**

#	Time	Action	Message	Duration / Fetch
8	21:47:43	CREATE TABLE EMPLOYEES ( employee_id INT AUTO_INCREMENT PRIMARY KEY, -- from getId) first_n...	0 row(s) affected	0.125 sec
9	21:49:49	CREATE TABLE CUSTOMERS ( customer_id INT AUTO_INCREMENT PRIMARY KEY, -- from getId) first_n...	0 row(s) affected	0.054 sec
10	21:50:09	CREATE TABLE SERVICES ( service_id INT AUTO_INCREMENT PRIMARY KEY, name VARCHAR(100) N...	0 row(s) affected	0.047 sec
11	21:50:47	CREATE TABLE PAYMENTS ( payment_id INT AUTO_INCREMENT PRIMARY KEY, -- from getId) amount ...	0 row(s) affected	0.047 sec
12	21:51:13	CREATE TABLE RESERVATIONS ( reservation_id INT AUTO_INCREMENT PRIMARY KEY, -- from getId) c...	0 row(s) affected	0.109 sec

**Object Info** **Session**

MySQL Workbench

DataBases - Warning - not su...

File Edit View Query Database Server Tools Scripting Help

Navigator

SCHEMAS

hotel\_le\_villa

Tables

- customers
- employees
- payments
- rooms
- services

Views

Stored Procedures

Administration Schemas

No object selected

Query 1

```

1 USE hotel_le_villa;
2 CREATE TABLE PAYMENTS (
3     payment_id INT AUTO_INCREMENT PRIMARY KEY, -- (from getId)
4     amount DECIMAL(10, 2) NOT NULL, -- (from getAmount)
5     payment_method VARCHAR(50) NOT NULL, -- (from getPaymentMethod)
6     payment_date DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP -- (from getDate)
7 );

```

SQLAdditions

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

Output

Action Output

#	Time	Action	Message	Duration / Fetch
7	21:47:21	CREATE TABLE ROOMS ( room_id INT AUTO_INCREMENT PRIMARY KEY, room_number VARCHAR(10) ...	Error Code: 1050. Table 'rooms' already exists	0.062 sec
8	21:47:43	CREATE TABLE EMPLOYEES ( employee_id INT AUTO_INCREMENT PRIMARY KEY, -- from getId) first_n...	0 row(s) affected	0.125 sec
9	21:49:49	CREATE TABLE CUSTOMERS ( customer_id INT AUTO_INCREMENT PRIMARY KEY, -- from getId) first_n...	0 row(s) affected	0.094 sec
10	21:50:09	CREATE TABLE SERVICES ( service_id INT AUTO_INCREMENT PRIMARY KEY, name VARCHAR(100) N...	0 row(s) affected	0.047 sec
11	21:50:47	CREATE TABLE PAYMENTS ( payment_id INT AUTO_INCREMENT PRIMARY KEY, -- from getId) amount ...	0 row(s) affected	0.047 sec

Object Info Session

MySQL Workbench

DataBases - Warning - not su...

File Edit View Query Database Server Tools Scripting Help

Navigator

SCHEMAS

hotel\_le\_villa

Tables

- customers
- employees
- payments
- rooms
- services

Views

Stored Procedures

Functions

sys

Administration Schemas

No object selected

Query 1

```

1 USE hotel_le_villa;
2 CREATE TABLE SERVICES (
3     service_id INT AUTO_INCREMENT PRIMARY KEY,
4     name VARCHAR(100) NOT NULL, -- e.g., 'Breakfast', 'Parking', 'Spa Access', 'Laundry'
5     cost DECIMAL(10, 2) NOT NULL,
6     description TEXT
7 );

```

SQLAdditions

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

Output

Action Output

#	Time	Action	Message	Duration / Fetch
5	21:33:49	DROP TABLE clientes	0 row(s) affected	0.171 sec
6	21:42:22	CREATE TABLE ROOMS ( room_id INT AUTO_INCREMENT PRIMARY KEY, room_number VARCHAR(10) ...	0 row(s) affected	0.250 sec
7	21:47:21	CREATE TABLE ROOMS ( room_id INT AUTO_INCREMENT PRIMARY KEY, room_number VARCHAR(10) ...	Error Code: 1050. Table 'rooms' already exists	0.062 sec
8	21:47:43	CREATE TABLE EMPLOYEES ( employee_id INT AUTO_INCREMENT PRIMARY KEY, -- from getId) first_n...	0 row(s) affected	0.125 sec
9	21:49:49	CREATE TABLE CUSTOMERS ( customer_id INT AUTO_INCREMENT PRIMARY KEY, -- from getId) first_n...	0 row(s) affected	0.094 sec

Object Info Session

```

USE hotel_le_villa;
CREATE TABLE CUSTOMERS (
    customer_id INT AUTO_INCREMENT PRIMARY KEY, -- (from getid)
    first_name VARCHAR(50) NOT NULL, -- (from getName)
    second_name VARCHAR(50), -- (from getSecondName)
    last_name VARCHAR(50) NOT NULL, -- (from getLastname)
    second_last_name VARCHAR(50), -- (from getSecondLastName)
    phone VARCHAR(20), -- (from getPhone)
    email VARCHAR(100) NOT NULL UNIQUE, -- (from getEmail)
    state VARCHAR(50), -- (from getState)
    curp VARCHAR(18) UNIQUE, -- (from getCurp)
    password_hash VARCHAR(255) -- (from getPassword)
);

```

Output

Action	Time	Message	Duration / Fetch
4 21:22:49 CREATE TABLE Clientes (		0 row(s) affected	0.094 sec
5 21:33:49 DROP TABLE clientes		0 row(s) affected	0.171 sec
6 21:42:22 CREATE TABLE ROOMS (		0 row(s) affected	0.250 sec
7 21:47:21 CREATE TABLE ROOMS (		Error Code: 1050. Table 'rooms' already exists	0.062 sec
8 21:47:43 CREATE TABLE EMPLOYEES (		0 row(s) affected	0.125 sec

## II. Documentación de la Clase DBConnection Completa

La documentación de la clase DBConnection se enriquece con el contrato de la interfaz y detalles críticos sobre seguridad y manejo de errores.

### 3. Clase DBConnection (en db\_connection.py)

#### A. Patrones y Propósito

- Patrón Pool de Conexiones:** Utiliza `mysql.connector.pooling.MySQLConnectionPool` para gestionar un conjunto de conexiones abiertas. Esto mitiga la latencia inherente a la apertura y cierre de conexiones, optimizando el rendimiento en entornos de alta concurrencia.
- Patrón Singleton:** El objeto `_db_connection_instance` se inicializa una sola vez a nivel de módulo al cargar `db_connection.py`. Esto **centraliza** la administración del pool, asegurando que solo exista un punto de acceso a la base de datos por proceso de aplicación, lo cual es crítico para la gestión de recursos.

#### B. Inyección de Configuración y Seguridad

Parámetro	Fuente	Comentario Crítico
<b>HOST,</b> <b>USER,</b> <b>PASSWORD</b>	Importados desde <code>mysql_env.py</code>	Para el despliegue en producción, se debe asegurar que el archivo <code>mysql_env.py</code> cargue estos valores desde <b>Variables de Entorno</b> (e.g.,

		usando os.environ) y no las almacene en texto plano, manteniendo los secretos <b>externos</b> al código binario.
<b>POOL_SIZE</b>	Definido en mysql_env.py	El valor predeterminado recomendado es entre <b>5 y 10</b> conexiones. Este valor debe ser ajustado en función de la carga máxima esperada y la capacidad del servidor de base de datos.
<b>Seguridad de la Conexión</b>	No mencionada	<b>Recomendación:</b> La configuración debe incluir soporte para <b>SSL/TLS</b> si la conexión a la base de datos es a través de una red no confiable, asegurando que la información de la aplicación esté cifrada en tránsito.

### C. Interfaz Pública y Manejo de Errores

Las funciones actúan como el contrato de servicio para todas las clases DAO.

Función	Descripción
get_conn()	<b>Obtiene y Retorna</b> un objeto de conexión (CMySQLConnection) del pool. Si el pool está agotado, la llamada bloqueará hasta que una conexión esté disponible.
close_conn(connection)	<b>Liberá</b> el objeto connection y lo devuelve al pool. Es una <b>responsabilidad obligatoria</b> de la clase DAO llamar a esta función, idealmente en una cláusula <b>finally</b> para garantizar la liberación de recursos.
<b>Manejo de Excepciones</b>	La inicialización del pool incluye un bloque try...except para capturar fallos de conexión iniciales (e.g., mysql.connector.errors.InterfaceError). En caso de fallo, la aplicación debe <b>terminar inmediatamente</b> o pasar a un modo de degradación (si aplica), ya que la infraestructura esencial no está disponible.

### Pruebas Unitarias

El objetivo de estas pruebas unitarias es garantizar dos aspectos cruciales del módulo de conexión a la base de datos:

1. **Aislamiento:** Usar **Mocking** para simular la librería `mysql.connector.pooling` y evitar cualquier conexión real o dependencia externa durante la ejecución de la prueba.
2. **Integridad del Patrón:** Verificar que el patrón **Singleton** se mantenga, asegurando que el *Pool de Conexiones* (`MySQLConnectionPool`) se inicialice **exactamente una vez**, independientemente de cuántas veces se solicite una conexión.
3. **Contrato Público:** Validar que los métodos públicos (`get_conn`, `close_conn`) interactúen correctamente con el pool simulado.

## Configuración del Entorno de Pruebas

Para ejecutar las pruebas se requiere un *framework* de pruebas (**pytest**) y una librería de *mocking* (**pytest-mock**).

## Requisitos de Entorno

Bash

```
# Instalación del framework de pruebas y la librería de mocking
pip install pytest pytest-mock
```

## Implementación de Pruebas con pytest y mocker

El módulo `pytest-mock` proporciona el *fixture* `mocker`, que permite sustituir objetos, funciones o clases reales por simulaciones controladas (`MagicMock`).

### `tests/test_db_connection.py`

```
import pytest
from unittest.mock import MagicMock
from db_connection import get_conn, close_conn, DBConnection
from mysql.connector.errors import InterfaceError


@pytest.fixture(autouse=True)
def mock_db_pool(mocker):
    """
    Fixture que simula (mokea) la clase MySQLConnectionPool de la libreria
    externa.

    Se parchea el objeto real para aislar la prueba, retornando un objeto Mock
    que simula el comportamiento de un Pool y la Conexion.
    """
    pass
```

```

MockPool = mocker.MagicMock()
MockConnection = mocker.MagicMock()
MockPool.return_value.get_connection.return_value = MockConnection

# Este es el mock que representa la CLASE MySQLConnectionPool
mock_class = mocker.patch(
    'db_connection.MySQLConnectionPool',
    return_value=MockPool.return_value
)

# Asegura que cada prueba comience con un estado limpio, reseteando el
Singleton.
DBConnection._instance = None

return mock_class

def test_singleton_pattern_pool_initialization(mock_db_pool):
    """
    Verifica el patron Singleton: el pool de conexiones solo debe ser
    inicializado una vez.

    La aserción clave es que la clase MySQLConnectionPool simulada
    (mock_db_pool)
    haya sido llamada para su construccion una sola vez, incluso tras
    multiples
    llamadas a la funcion publica get_conn().
    """
    conn1 = get_conn()

    conn2 = get_conn()

    mock_db_pool.assert_called_once() # Verifica que el constructor fue
    llamado una vez
    assert mock_db_pool.return_value.get_connection.call_count == 2


def test_connection_is_returned_to_pool(mock_db_pool):
    """
    Verifica el contrato de close_conn(): Al cerrar la conexion, se debe
    llamar
    al metodo .close() de la conexion prestada, devolviendola al pool.
    """
    mock_conn = get_conn()
    close_conn(mock_conn)
    mock_conn.close.assert_called_once()

```

```

def test_initialization_failure_raises_exception(mocker):
    """
        Prueba que si la inicialización del Pool falla (ej. DB caida o
        credenciales invalidas),
        la excepcion sea propagada o manejada correctamente.

        Se simula que el constructor de MySQLConnectionPool lanza un
        InterfaceError.

    """
    # Asegura un estado limpio para esta prueba también.
    DBConnection._instance = None

    mocker.patch(
        'db_connection.MySQLConnectionPool',
        side_effect=InterfaceError("Host caido o credenciales invalidas")
    )

    with pytest.raises(InterfaceError):
        get_conn()

```

```

import unittest
from unittest.mock import MagicMock, patch
import mysql.connector

# Importar las clases de los componentes a probar
from customer import Customer
from payment import Payment
from reservationService import ServiceReservation
from reservation import Reservation
from dao.customer_dao import CustomerDAO # Asegúrate que la ruta al DAO es
correcta
from dao.reservation_dao import ReservationDAO

# -----
# I. Pruebas de la Capa DAO (Persistencia y Transacciones)
# -----


# Usamos @patch para simular las funciones de conexión del módulo
db_connection
@patch('dao.customer_dao.get_conn')
@patch('dao.customer_dao.close_conn')
class TestCustomerDAOLogic(unittest.TestCase):

    # SETUP de MOCK para la conexión y el cursor

```

```

def setUp(self):
    self.dao = CustomerDAO()
    self.mock_customer = Customer(0, "Test", "S", "User", "A", "123",
"test@mail.com", "State", "CURP123", "hash")

# TEST 1: Validación de creación exitosa y Mapeo Objeto-Relacional (ORM)
def test_01_create_customer_success(self, mock_close_conn, mock_get_conn):
    mock_conn = MagicMock()
    mock_cursor = MagicMock()
    mock_cursor.lastrowid = 42
    mock_conn.cursor.return_value = mock_cursor
    mock_get_conn.return_value = mock_conn
    new_id = self.dao.create(self.mock_customer)
    self.assertEqual(new_id, 42)
    self.assertEqual(self.mock_customer.getId(), 42)
    mock_cursor.execute.assert_called_once()
    mock_conn.commit.assert_called_once()

# TEST 2: Manejo de Excepciones de la BD (Transaccionalidad)
def test_02_create_customer_db_error_handles_rollback(self,
mock_close_conn, mock_get_conn):
    mock_conn = MagicMock()
    mock_cursor = MagicMock()
    mock_cursor.execute.side_effect = mysql.connector.Error("Simulated DB
Error")
    mock_conn.cursor.return_value = mock_cursor
    mock_get_conn.return_value = mock_conn
    new_id = self.dao.create(self.mock_customer)
    self.assertIsNone(new_id)
    mock_conn.rollback.assert_called_once()
    self.assertEqual(self.mock_customer.getId(), 0)

# Pruebas de Entidades de Dominio (Lógica de Negocio y Colaboración)

class TestReservationLogic(unittest.TestCase):
    def setUp(self):
        self.mock_room = MagicMock()
        self.mock_customer = MagicMock()
        self.reservation = Reservation(1, "2025-01-01", "2025-01-05",
self.mock_customer, self.mock_room)

#3 Validación de Transición de Estado y Colaboración
def test_03_create_reservation_success_collaboration(self):
    self.mock_room.getStatus.return_value = "Available"
    success = self.reservation.createReservation()
    self.assertTrue(success)

```

```
self.mock_room.assignCustomer.assert_called_once_with(self.mock_customer)

self.mock_customer.makeReservation.assert_called_once_with(self.reservation)
def test_04_payment_encapsulation_and_date_assignment(self):
    with patch('payment.date') as mock_date:
        mock_date.today.return_value = '2025-11-26'
        pay = Payment(100, 500.50, "Credit Card")
        pay.setAmount(600.75)
        self.assertEqual(pay.getAmount(), 600.75)
        self.assertEqual(pay.getDate(), '2025-11-26')

# 4 Validación de Lógica de Colecciones
def test_05_customer_reservation_list_management(self):
    cust = Customer(1, "Test", "", "User", "", "123", "e@mail.com",
"State", "CURP123")
    mock_res_1 = MagicMock()
    mock_res_1.getId.return_value = 101
    cust.makeReservation(mock_res_1)
    self.assertEqual(len(cust._Customer__reservations), 1)
    success = cust.cancelReservation(101)
    self.assertTrue(success)
    self.assertEqual(len(cust._Customer__reservations), 0)
    fail = cust.cancelReservation(999)
    self.assertFalse(fail)

if __name__ == '__main__':
    unittest.main()
```

## Ejecución de las Pruebas

Ejecuta las pruebas desde la raíz de tu proyecto usando el comando `pytest`.

### Comando de Ejecución

Bash

`pytest`

```
----- 3 passed in 0.20s -----
(vENV) PS C:\Users\andlo\Desktop\proyectPartial2_optativa> pytest
>>
===== test session starts =====
platform win32 -- Python 3.13.9, pytest-9.0.1, pluggy-1.6.0
rootdir: C:\Users\andlo\Desktop\proyectPartial2_optativa
configfile: pytest.ini
plugins: mock-3.15.1
collected 8 items

test\test_db_connection.py ...
test\test_other.py ......

===== 8 passed in 0.64s =====
(vENV) PS C:\Users\andlo\Desktop\proyectPartial2_optativa>
```

## Control de versiones con GITHUB

### Documentación Técnica de 4 Commits (Registro Secuencial Detallado)

Este enfoque registra la construcción del sistema en cuatro hitos técnicos clave, separando claramente la infraestructura, la persistencia y la validación.

#### Commit 1/4: refactor(infra): Implementación del Pool de Conexiones y externalización de credenciales.

Se refactorizó el módulo **db\_connection.py** para integrar el patrón **Pool de Conexiones** de MySQL, optimizando la gestión de recursos de la BD y reduciendo la latencia de conexión. Se creó **mysql\_env.py** para desacoplar las credenciales de la BD del código binario, cargándolas desde variables de entorno para una gestión segura de la configuración. Se añadió el manejo robusto de excepciones críticas al inicializar la conexión.

#### Commit 2/4: feat(dao): Creación de la Capa de Acceso a Datos (DAO) inicial.

Se inició la capa de persistencia con la creación de la estructura de directorio **dao/**. Se implementaron los objetos **ServiceDAO** y **employee\_dao**, definiendo los contratos de métodos **CRUD** para los servicios y el personal. Esto establece la **Separación de Responsabilidades** entre la lógica de dominio y las operaciones de la BD.

#### Commit 3/4: feat(dao): Finalización del mapeo ORM y script de inicialización de datos.

Se completó la capa DAO con la adición de **customer\_dao.py**, **room\_dao.py**, **payment\_dao.py** y **reservation\_dao.py**. Todos los DAOs implementan la lógica de **Mapeo Objeto-Relacional (ORM) manual** para transformar registros de la BD en

instancias de entidades de dominio. Se introdujo `populate_db.py` para la inicialización controlada de datos de prueba en las tablas.

#### **Commit 4/4: test(unit): Implementación de las 5 pruebas unitarias con validación de transacciones y lógica de dominio.**

Se completó el módulo de pruebas unitarias para validar los componentes críticos del sistema. Se incluyen pruebas *mockeadas* para:

1. Verificar la **integridad transaccional** del DAO, asegurando el `conn.rollback()` ante `mysql.connector.Error`.
2. Validar el **Mapeo ORM** (asignación de `cursor.lastrowid` a `entity.setId()`).
3. Confirmar la **colaboración de objetos** en `Reservation.createReservation()`.
4. Probar la lógica de **colecciones internas** de la entidad Customer.
5. Asegurar el **Encapsulamiento** de la entidad Payment.

#### **Documentación Técnica de 3 Commits (Con Manejo de Ramas)**

Este esquema utiliza una **rama de característica** (`feature/qa`) para el desarrollo de pruebas, registrando el avance del código funcional en `main` y la validación en una rama aislada.

#### **Flujo de Trabajo**

1. **Commit en la rama main**
2. **git checkout -b feature/qa** (Creación de la rama de calidad)
3. **Commit en la rama feature/qa**
4. **git checkout main**
5. **git merge feature/qa** (Fusión de la rama)

#### **Commit 1/3: feat(arch): Implementación total de la capa de persistencia y configuración de Pool.**

Se estableció la arquitectura de persistencia completa. Esto incluye: configuración del **Pool de Conexiones** en la infraestructura, creación de la carpeta `dao/` y definición de todos los DAOs (CustomerDAO, ReservationDAO, RoomDAO, etc.). Este commit consolida todo el código funcional de base de datos listo para ser probado.

#### **Commit 2/3 (en rama feature/qa): test(all): Desarrollo y ejecución de 5 pruebas unitarias con mocking y cobertura de excepciones.**

En la rama de calidad, se implementó el módulo de *tests unitarios* utilizando `unittest.mock` para aislar los DAOs y las entidades. Se cubren las pruebas de

**rollback transaccional**, el correcto **Mapeo ORM** y la **Lógica de negocio** de colaboración entre objetos de dominio (Customer, Reservation).

**Commit 3/3 (Merge a main): merge(qa): Fusión de feature/qa: Código validado con pruebas unitarias.**

Se integró la rama feature/qa a la rama principal (main). Este commit atestigua que el código funcional introducido anteriormente ha superado la validación de las pruebas unitarias y ahora forma parte de la línea de desarrollo estable.

**Documentación Técnica de 2 Commits (Bloques Funcionales Suficientes)**

Este esquema registra el trabajo en dos fases principales: la implementación completa del *backend* de datos y la certificación de calidad.

**Commit 1/2: feat(core): Sistema de persistencia funcional completo y Pool de Conexiones.**

Se completó el desarrollo de la arquitectura de la base de datos y la capa de persistencia. Se configuró el **Pool de Conexiones** para un manejo eficiente de recursos y se crearon la totalidad de los objetos **DAO** para todas las entidades del dominio. Este commit asegura que el sistema puede realizar todas las operaciones CRUD necesarias y mapear los datos a objetos en memoria.

**Commit 2/2: ci(test): Módulo de pruebas unitarias implementado; verificación de transacciones y ORM.**

Se implementó y ejecutó el módulo de pruebas unitarias cubriendo los **5 requisitos mínimos**. Las pruebas validan la **integridad transaccional** (manejo de rollback en los DAOs), la correcta actualización del ID generado por la BD (lastrowid), y la validación de la **colaboración entre objetos** de negocio para la gestión de recursos (p. ej., asignación de habitaciones).