COMPUTER PROGRAMMING 2

M001

FINAL COURSE OUTPUT PORTFOLIO

SUBMITTED BY:
CARL VICTOR A. BUENAFE

SUBMITTED TO:
DEAN RODRIGO BELLEZA JR.

FINAL COURSE OUTCOMES:

DEVELOPING A PYTHON PROGRAM WITH GUI AND DATABASE MANIPULATION, STUDENTS DEMONSTRATE THEIR ABILITY TO BUILD PRACTICAL APPLICATIONS THAT SOLVE REAL-WORLD PROBLEMS. THIS PREPARES THEM FOR FURTHER STUDIES OR CAREERS IN SOFTWARE DEVELOPMENT, DATA SCIENCE, OR OTHER FIELDS THAT INVOLVE WORKING WITH DATA AND USER INTERFACES.

PERFORMANCE INDICATOR

CONSTRUCT AN OBJECT-ORIENTED PYTHON PROGRAM THAT WILL USE PYQT6 TO BUILD ITS USER INTERFACE WITH MINIMUM OF 4 FORMS INCLUDING MAIN MENU AND EXECUTES CRUD (CREATE, RETRIEVE, UPDATE, DELETE) OPERATIONS WITHIN A MYSQL DATABASE WITH MINIMUM OF 3 TABLES UTILIZING STRUCTURED QUERY LANGUAGE (SQL).

SYSTEM DESCRIPTION

THE SYSTEM DESCRIPTION CALLS FOR THE DEVELOPMENT OF A PYTHON PROGRAM CAPABLE OF PERFORMING CRUD OPERATIONS (CREATE, RETRIEVE, UPDATE, DELETE) ON A DATABASE USING SQL. THIS INVOLVES CONNECTING TO THE DATABASE, DEFINING FUNCTIONS FOR EACH CRUD OPERATION, EXECUTING SQL STATEMENTS WITHIN THESE FUNCTIONS, HANDLING ERRORS, AND TESTING THE PROGRAM TO ENSURE ITS FUNCTIONALITY. THIS PYTHON-BASED SYSTEM SIMPLIFIES DATABASE MANAGEMENT BY ALLOWING USERS TO PERFORM CRUD OPERATIONS (CREATE, RETRIEVE, UPDATE, DELETE) THROUGH A USER-FRIENDLY INTERFACE. IT CONNECTS TO THE CHOSEN DATABASE (E.G., MYSQL, POSTGRESQL, SQLITE) AND USES SQL COMMANDS TO MANIPULATE DATA. KEY FEATURES INCLUDE FUNCTIONS FOR EACH CRUD OPERATION, ERROR HANDLING TO ENSURE DATA INTEGRITY, AND THE POTENTIAL FOR A COMMAND-LINE OR GRAPHICAL USER INTERFACE. THE SYSTEM OFFERS FLEXIBILITY FOR DIFFERENT DATABASES, STREAMLINES DATA TASKS, AND CAN BE USED IN VARIOUS APPLICATIONS LIKE CUSTOMER MANAGEMENT, INVENTORY TRACKING, OR EDUCATION.

LIST OF TABLES

```
CREATE TABLE IF NOT EXISTS CUSTOMERS (
    CUSTOMER_ID INT AUTO_INCREMENT PRIMARY KEY,
    NAME VARCHAR(255) NOT NULL,
    EMAIL VARCHAR(255),
    CITY VARCHAR(255)
);

CREATE TABLE IF NOT EXISTS ROOMS (
    ROOM_ID INT AUTO_INCREMENT PRIMARY KEY,
    CUSTOMER_ID INT,
    ROOM_NO INT,
    FOREIGN KEY (CUSTOMER_ID) REFERENCES CUSTOMERS(CUSTOMER_ID) ON DELETE SET NULL
);

CREATE TABLE IF NOT EXISTS BOOKINGS (
    BOOKING_ID INT AUTO_INCREMENT PRIMARY KEY,
    CUSTOMER_ID INT,
    ROOM_ID INT,
    DATE_BOOKED DATE,
    CHECK_IN_DATE DATE,  -- ADD CHECK-IN DATE COLUMN
    CHECK_OUT_DATE DATE, -- ADD CHECK-OUT DATE COLUMN
    FOREIGN KEY (CUSTOMER_ID) REFERENCES CUSTOMERS(CUSTOMER_ID) ON DELETE CASCADE,
    FOREIGN KEY (ROOM_ID) REFERENCES ROOMS(ROOM_ID) ON DELETE CASCADE
);
```

SQL

```
CREATE DATABASE IF NOT EXISTS HOTELDATABASE;

DROP DATABASE HOTELDATABASE;

USE HOTELDATABASE;

CREATE TABLE IF NOT EXISTS CUSTOMERS (
    CUSTOMER_ID INT AUTO_INCREMENT PRIMARY KEY,
    NAME VARCHAR(255) NOT NULL,
    EMAIL VARCHAR(255),
    CITY VARCHAR(255)
);

CREATE TABLE IF NOT EXISTS ROOMS (
    ROOM_ID INT AUTO_INCREMENT PRIMARY KEY,
    CUSTOMER_ID INT,
    ROOM_NO INT,
     FOREIGN KEY (CUSTOMER_ID) REFERENCES CUSTOMERS(CUSTOMER_ID)
ON DELETE SET NULL
);

CREATE TABLE IF NOT EXISTS BOOKINGS (
    BOOKING_ID INT AUTO_INCREMENT PRIMARY KEY,
    CUSTOMER_ID INT,
    ROOM_ID INT,
    DATE_BOOKED DATE,
    CHECK_IN_DATE DATE,  -- ADD CHECK-IN DATE COLUMN
    CHECK_OUT_DATE DATE, -- ADD CHECK-OUT DATE COLUMN
     FOREIGN KEY (CUSTOMER_ID) REFERENCES CUSTOMERS(CUSTOMER_ID)
ON DELETE CASCADE,
      FOREIGN KEY (ROOM_ID) REFERENCES ROOMS(ROOM_ID) ON DELETE
CASCADE
);

-- UPDATE COMMANDS
UPDATE CUSTOMERS
SET EMAIL = 'NEWEMAIL@EXAMPLE.COM'
WHERE CUSTOMER_ID = 1;  -- UPDATE THE EMAIL OF CUSTOMER WITH ID 1

UPDATE ROOMS
SET ROOM_NO = 202
WHERE ROOM_ID = 3;     -- UPDATE THE ROOM NUMBER OF ROOM WITH ID 3

-- DELETE COMMANDS
DELETE FROM ROOMS
WHERE ROOM_ID = 5;     -- DELETE THE ROOM WITH ID 5

DELETE FROM CUSTOMERS
WHERE CUSTOMER_ID = 2;  -- DELETE THE CUSTOMER WITH ID 2 (THIS WILL
ALSO DELETE THEIR ASSOCIATED ROOMS DUE TO THE ON DELETE CASCADE
CONSTRAINT)
```

COMPLETE PYTHON SOURCE CODE

```
import sys
from PyQt6.QtWidgets import (
    QApplication, QMainWindow, QWidget, QVBoxLayout, QTableWidget,
QTableWidgetItem,
    QPushButton, QHBoxLayout, QLineEdit, QMessageBox, QComboBox, QDialogButtonBox,
QLabel, QDialog, QDateEdit,
)
from PyQt6.QtCore import QDate
import mysql.connector

# Database Connection
db = mysql.connector.connect(
    host="localhost",
    user="root",
    password="civic328",
    database="hoteldatabase"
)
db_config = {
    'host': 'localhost',
    'user': 'root',
    'password': 'civic328',
    'database': 'hoteldatabase'
}
cursor = db.cursor()

class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Buenafe - Hotel Room Reservation")
        self.setGeometry(100, 100, 800, 600)

        # Connect to MySQL database
        self.db = mysql.connector.connect(**db_config)
        self.cursor = self.db.cursor()

        # Create main layout
        central_widget = QWidget()
        main_layout = QVBoxLayout()
        central_widget.setLayout(main_layout)
        self.setCentralWidget(central_widget)
        self.customer_table.setRowCount(0)
```

COMPLETE PYTHON SOURCE CODE

```
# Create buttons for managing customers and rooms
button_layout = QHBoxLayout()
manage_customers_button = QPushButton("Manage Customers")
manage_customers_button.clicked.connect(self.show_manage_customers_window)
button_layout.addWidget(manage_customers_button)
manage_rooms_button = QPushButton("Manage Book Rooms")
manage_rooms_button.clicked.connect(self.show_manage_rooms_window)
button_layout.addWidget(manage_rooms_button)
main_layout.addLayout(button_layout)

#Add button for manage booking
manage_bookings_button = QPushButton("Manage Book Dates")
manage_bookings_button.clicked.connect(self.show_manage_booked_dates_window)
button_layout.addWidget(manage_bookings_button)
main_layout.addLayout(button_layout)


def show_manage_customers_window(self):
self.manage_customers_window = ManageCustomersWindow()
self.manage_customers_window.show()

def show_manage_rooms_window(self):
self.manage_rooms_window = ManageRoomsWindow(db)
self.manage_rooms_window.show()

def show_manage_booked_dates_window(self):
self.manage_booked_dates_window = ManageBookDatesWindow(self.db, self.cursor)
self.manage_booked_dates_window.show()

class ManageCustomersWindow(QWidget):
def __init__(self):
super().__init__()
self.setWindowTitle("Buenafe - Manage Customers")
self.setGeometry(100, 100, 800, 600)

# Create main layout
main_layout = QVBoxLayout()
self.setLayout(main_layout)

# Create table for customers
self.customer_table = QTableWidget()
self.customer_table.setColumnCount(4)
self.customer_table.setHorizontalHeaderLabels(["Customer ID", "Name", "Email",
"City/Municipality"])
self.load_customer_data()
main_layout.addWidget(self.customer_table
```

COMPLETE PYTHON SOURCE CODE

```
# Create buttons for managing customers
        button_layout = QHBoxLayout()
        add_customer_button = QPushButton("Add Customer")
        add_customer_button.clicked.connect(self.add_customer)
        button_layout.addWidget(add_customer_button)
        edit_customer_button = QPushButton("Edit Customer")
        edit_customer_button.clicked.connect(self.edit_customer)
        button_layout.addWidget(edit_customer_button)
        delete_customer_button = QPushButton("Delete Customer")
        delete_customer_button.clicked.connect(self.delete_customer)
        button_layout.addWidget(delete_customer_button)
        main_layout.addLayout(button_layout)

    def load_customer_data(self):
        # Clear the table
        self.customer_table.setRowCount(0)  # Remove existing rows

        try:
            # Fetch customer data from the database
            cursor.execute("SELECT * FROM customers")
            customers = cursor.fetchall()

            # Populate the table with customer data
            for customer in customers:
                row_position = self.customer_table.rowCount()
                self.customer_table.insertRow(row_position)

                # Create table items for each column
                for col_idx, data in enumerate(customer):
                    item = QTableWidgetItem(str(data))
                    self.customer_table.setItem(row_position, col_idx, item)

        except mysql.connector.Error as err:
            QMessageBox.critical(self, "Error", f"Failed to load customer data: {err}")

    def add_customer(self):
        dialog = AddCustomerDialog(self)  # Create the dialog
        if dialog.exec() == QDialog.DialogCode.Accepted:
            name, email, city = dialog.get_customer_data()

            # Basic input validation
            if not name or not email or not city:
                QMessageBox.warning(self, "Error", "Please fill in all fields.")
                return
```

COMPLETE PYTHON SOURCE CODE

```
try:
# Insert the new customer into the database
cursor.execute(
"INSERT INTO customers (name, email, city) VALUES (%s, %s, %s)",
(name, email, city) # Pass values as a tuple for parameterization
)
db.commit()

# Refresh the customer table
self.load_customer_data()

QMessageBox.information(self, "Success", "Customer added successfully!")
except mysql.connector.Error as err:
QMessageBox.critical(self, "Error", f"Failed to add customer: {err}")

def edit_customer(self):
# Get the selected customer from the table
selected_row = self.customer_table.currentRow()
if selected_row == -1: # No row selected
QMessageBox.warning(self, "Error", "Please select a customer to edit.")
return

# Get customer details from the selected row
customer_id = int(self.customer_table.item(selected_row, 0).text())
name = self.customer_table.item(selected_row, 1).text()
email = self.customer_table.item(selected_row, 2).text()
city = self.customer_table.item(selected_row, 3).text()

# Create the EditCustomerDialog
dialog = EditCustomerDialog(customer_id, name, email, city, self)

if dialog.exec() == QDialog.DialogCode.Accepted:
# Get the updated customer data
updated_name, updated_email, updated_city = dialog.get_customer_data()

# Basic input validation
if not updated_name or not updated_email or not updated_city:
QMessageBox.warning(self, "Error", "Please fill in all fields.")
return

try:
# Update the customer in the database
cursor.execute(
"UPDATE customers SET name = %s, email = %s, city = %s WHERE customer_id = %s",
(updated_name, updated_email, updated_city, customer_id)
)
db.commit()

# Refresh the customer table
self.load_customer_data()

QMessageBox.information(self, "Success", "Customer updated successfully!")
except mysql.connector.Error as err:
QMessageBox.critical(self, "Error", f"Failed to update customer: {err}")
```

COMPLETE PYTHON SOURCE CODE

```
def delete_customer(self):
    # Get the selected customer from the table
    selected_row = self.customer_table.currentRow()
    if selected_row == -1:  # No row selected
        QMessageBox.warning(self, "Error", "Please select a customer to delete.")
        return

    # Get the customer ID to delete
    customer_id = int(self.customer_table.item(selected_row, 0).text())  # Assuming
customer ID is in the first column

    # Confirmation dialog
    confirm = QMessageBox.question(
        self,
        "Confirm Deletion",
        "Are you sure you want to delete this customer?",
        QMessageBox.StandardButton.Yes | QMessageBox.StandardButton.No
    )

    if confirm == QMessageBox.StandardButton.Yes:
        try:
            # Delete the customer from the database
            cursor.execute("DELETE FROM customers WHERE customer_id = %s",
(customer_id,))
            db.commit()

            # Delete associated rooms if needed (add this if you have foreign key
constraints)
            # cursor.execute("DELETE FROM rooms WHERE customer_id = %s",
(customer_id,))
            # db.commit()

            # Refresh the customer table
            self.load_customer_data()

            QMessageBox.information(self, "Success", "Customer deleted successfully!")
        except mysql.connector.Error as err:
            QMessageBox.critical(self, "Error", f"Failed to delete customer: {err}")

# Dialog classes for adding and editing customers
class AddCustomerDialog(QDialog):
    def __init__(self, parent=None):
        super().__init__(parent)
        self.setWindowTitle("Buenafe - Add Customer")
```

COMPLETE PYTHON SOURCE CODE

```
# Create input fields for customer data
name_layout = QHBoxLayout()
name_layout.addWidget(QLabel("Name:"))
self.name_input = QLineEdit()
name_layout.addWidget(self.name_input)

email_layout = QHBoxLayout()
email_layout.addWidget(QLabel("Email:"))
self.email_input = QLineEdit()
email_layout.addWidget(self.email_input)

city_layout = QHBoxLayout()
city_layout.addWidget(QLabel("City/Municipality:"))
self.city_input = QLineEdit()
city_layout.addWidget(self.city_input)

# Add input fields to the dialog
layout = QVBoxLayout()
layout.addLayout(name_layout)
layout.addLayout(email_layout)
layout.addLayout(city_layout)

# Add buttons to the dialog
button_box = QDialogButtonBox(QDialogButtonBox.StandardButton.Ok | QDialogButtonBox.StandardButton.Cancel)
button_box.accepted.connect(self.accept)
button_box.rejected.connect(self.reject)
layout.addWidget(button_box)

self.setLayout(layout)

def get_customer_data(self):
return self.name_input.text(), self.email_input.text(), self.city_input.text()

class EditCustomerDialog(QDialog):
def __init__(self, customer_id, name, email, city, parent=None):
super().__init__(parent)
self.setWindowTitle("Buenafe - Edit Customer")
self.customer_id = customer_id

# Create input fields for customer data (pre-filled with existing data)
name_layout = QHBoxLayout()
name_layout.addWidget(QLabel("Name:"))
self.name_input = QLineEdit(name)
name_layout.addWidget(self.name_input)

email_layout = QHBoxLayout()
email_layout.addWidget(QLabel("Email:"))
self.email_input = QLineEdit(email)
email_layout.addWidget(self.email_input)

city_layout = QHBoxLayout()
city_layout.addWidget(QLabel("City/Municipality:"))
self.city_input = QLineEdit(city)
city_layout.addWidget(self.city_input)

# Add input fields to the dialog
layout = QVBoxLayout()
layout.addLayout(name_layout)
layout.addLayout(email_layout)
layout.addLayout(city_layout)
```

COMPLETE PYTHON SOURCE CODE

```
button_box = QDialogButtonBox(QDialogButtonBox.StandardButton.Ok |
QDialogButtonBox.StandardButton.Cancel)
        button_box.accepted.connect(self.accept)
        button_box.rejected.connect(self.reject)
        layout.addWidget(button_box)

        self.setLayout(layout)

        def accept(self):
            new_name = self.name_input.text()
            new_email = self.email_input.text()
            new_city = self.city_input.text()

            # Input validation (you should add more robust validation)
            if not new_name or not new_email or not new_city:
                QMessageBox.warning(self, "Error", "Please fill in all fields.")
                return

            try:
                # Update the customer in the database
                cursor.execute(
                    "UPDATE customers SET name = %s, email = %s, city = %s WHERE customer_id = %s",
                    (new_name, new_email, new_city, self.customer_id)
                )
                db.commit()

                QMessageBox.information(self, "Success", "Customer updated successfully!")
            except mysql.connector.Error as err:
                QMessageBox.critical(self, "Error", f"Failed to update customer: {err}")

            # Call the base class accept method to close the dialog
            super().accept()


    def get_customer_data(self):
        return self.name_input.text(), self.email_input.text(), self.city_input.text()


class ManageRoomsWindow(QWidget):
    def __init__(self, db):
        super().__init__()
        self.db = db
        self.cursor = db.cursor()
        self.setWindowTitle("Buenafe - Manage Book Rooms")
        self.setGeometry(100, 100, 800, 600)

        # Create main layout
        main_layout = QVBoxLayout()
        self.setLayout(main_layout)

        # Create table for rooms
        self.rooms_table = QTableWidget()
        self.rooms_table.setColumnCount(4)  # 4 columns for room_id, customer_id, name, room_no
        self.rooms_table.setHorizontalHeaderLabels(["Room ID", "Customer ID", "Name", "Room No."])
        self.load_rooms_data()
        main_layout.addWidget(self.rooms_table)
```

COMPLETE PYTHON SOURCE CODE

```
# Create buttons for managing rooms
button_layout = QHBoxLayout()
add_room_button = QPushButton("Add Room")
add_room_button.clicked.connect(self.add_room)
button_layout.addWidget(add_room_button)
edit_room_button = QPushButton("Edit Room")
edit_room_button.clicked.connect(self.edit_room)
button_layout.addWidget(edit_room_button)
delete_room_button = QPushButton("Delete Room")
delete_room_button.clicked.connect(self.delete_room)
button_layout.addWidget(delete_room_button)
main_layout.addLayout(button_layout)

def load_rooms_data(self):
self.rooms_table.setRowCount(0)
try:
self.cursor.execute(
"SELECT r.room_id, c.customer_id, c.name, r.room_no "
"FROM rooms r LEFT JOIN customers c ON r.customer_id = c.customer_id"
)
rooms = self.cursor.fetchall()
for room in rooms:
row_position = self.rooms_table.rowCount()
self.rooms_table.insertRow(row_position)
for i, data in enumerate(room):
item = QTableWidgetItem(str(data) if data is not None else "")
self.rooms_table.setItem(row_position, i, item)
except mysql.connector.Error as err:
QMessageBox.critical(self, "Error", f"Failed to load room data: {err}")

def add_room(self):
dialog = AddRoomDialog(self, self.db)
if dialog.exec() == QDialog.DialogCode.Accepted:
customer_id, room_no = dialog.get_room_data()

# Input validation
if not customer_id or not room_no:
QMessageBox.warning(self, "Error", "Please fill in all fields.")
return

try:
room_no = int(room_no) # Ensure room number is an integer
```

COMPLETE PYTHON SOURCE CODE

```
self.cursor.execute("SELECT * FROM rooms WHERE room_no = %s", (room_no,))
            existing_room = self.cursor.fetchone()
            if existing_room:
                QMessageBox.warning(self, "Error", "Room number already exists.")
                return

            # Insert the new room into the database
            self.cursor.execute(
                "INSERT INTO rooms (customer_id, room_no) VALUES (%s, %s)", (customer_id,
room_no)
            )
            self.db.commit()

            # Refresh the rooms table
            self.load_rooms_data()

            QMessageBox.information(self, "Success", "Room added successfully!")

        except ValueError:  # Catch invalid room number input (non-integer)
            QMessageBox.warning(self, "Error", "Room number must be an integer.")
        except mysql.connector.Error as err:
            QMessageBox.critical(self, "Error", f"Failed to add room: {err}")

    def get_customer_id_by_name(self, name):
        if not name:  # Handle empty name
            return None

        try:
            self.cursor.execute(
                "SELECT customer_id FROM customers WHERE name = %s", (name,)
            )

            result = self.cursor.fetchone()
            return result[0] if result else None  # Extract the ID or return None
        except mysql.connector.Error as err:
            QMessageBox.critical(self, "Error", f"Database error: {err}")
            return None

    def edit_room(self):
        selected_row = self.rooms_table.currentRow()
        if selected_row == -1:
            QMessageBox.warning(self, "Error", "Please select a room to edit.")
            return
```

COMPLETE PYTHON SOURCE CODE

```
room_id = int(self.rooms_table.item(selected_row, 0).text())
customer_id = int(self.rooms_table.item(selected_row, 1).text()) # Get customer ID from table
room_no = int(self.rooms_table.item(selected_row, 3).text())

dialog = EditRoomDialog(room_id, customer_id, room_no, self.db)
if dialog.exec() == QDialog.DialogCode.Accepted:
self.load_rooms_data()

def delete_room(self):
selected_row = self.rooms_table.currentRow()
if selected_row == -1:
QMessageBox.warning(self, "Error", "Please select a room to delete.")
return

# Get the room ID from the table
room_id = int(self.rooms_table.item(selected_row, 0).text()) # Assuming room_id is in the first column
(index 0)

# Ask for confirmation before deleting
confirm = QMessageBox.question(
self,
"Confirm Deletion",
"Are you sure you want to delete this room?",
QMessageBox.StandardButton.Yes | QMessageBox.StandardButton.No
)

if confirm == QMessageBox.StandardButton.Yes:
try:
# Delete the room from the database
self.cursor.execute("DELETE FROM rooms WHERE room_id = %s", (room_id,))
self.db.commit()

# Refresh the rooms table
self.load_rooms_data()

QMessageBox.information(self, "Success", "Room deleted successfully!")
except mysql.connector.Error as err:
QMessageBox.critical(self, "Error", f"Failed to delete room: {err}")

class AddRoomDialog(QDialog):
def __init__(self, parent=None, db=None):
super().__init__(parent)
self.db = db
self.setWindowTitle("Buenafe - Add Room")

room_no_layout = QHBoxLayout()
room_no_layout.addWidget(QLabel("Room No.:"))
self.room_no_input = QLineEdit()
room_no_layout.addWidget(self.room_no_input)

customer_id_layout = QHBoxLayout()
customer_id_layout.addWidget(QLabel("Customer ID:"))
self.customer_id_combo = QComboBox()
self.load_customer_ids()
customer_id_layout.addWidget(self.customer_id_combo)
```

COMPLETE PYTHON SOURCE CODE

```
layout = QVBoxLayout()
        layout.addLayout(room_no_layout)
        layout.addLayout(customer_id_layout)

        # Add buttons to the dialog
        button_box = QDialogButtonBox(QDialogButtonBox.StandardButton.Ok |
QDialogButtonBox.StandardButton.Cancel)
        button_box.accepted.connect(self.accept)
        button_box.rejected.connect(self.reject)
        layout.addWidget(button_box)

        self.setLayout(layout)

    def load_customer_ids(self):
        cursor = self.db.cursor() # Use the cursor from the instance variable
        cursor.execute("SELECT customer_id, name FROM customers")
        customers = cursor.fetchall()
        for customer in customers:
            self.customer_id_combo.addItem(f"{customer[0]} - {customer[1]}", customer[0])

    def get_room_data(self):
        customer_id = self.customer_id_combo.currentData()
        room_no = self.room_no_input.text()
        return customer_id, room_no


class EditRoomDialog(QDialog):
    def __init__(self, room_id, customer_id, room_no, db, parent=None):
        super().__init__(parent)
        self.setWindowTitle("Buenafe - Edit Room")
        self.room_id = room_id
        self.db = db
        self.cursor = db.cursor()

        # Customer ID Combo Box
        customer_id_layout = QHBoxLayout()
        customer_id_layout.addWidget(QLabel("Customer ID:"))
        self.customer_id_combo = QComboBox()
        self.load_customer_ids(customer_id)  # Load and set the current customer ID
        customer_id_layout.addWidget(self.customer_id_combo)

        # Room Number Input
        room_no_layout = QHBoxLayout()
        room_no_layout.addWidget(QLabel("Room No.:"))
        self.room_no_input = QLineEdit()
        self.room_no_input.setText(str(room_no))
        room_no_layout.addWidget(self.room_no_input)

        # Add input fields to the dialog
        layout = QVBoxLayout()
```

COMPLETE PYTHON SOURCE CODE

```
layout.addLayout(customer_id_layout)
layout.addLayout(room_no_layout)

# Add buttons using QDialogButtonBox
button_box = QDialogButtonBox(QDialogButtonBox.StandardButton.Ok |
QDialogButtonBox.StandardButton.Cancel)
button_box.accepted.connect(self.accept)
button_box.rejected.connect(self.reject)
layout.addWidget(button_box)

self.setLayout(layout)

def load_customer_ids(self, current_customer_id=None):
self.cursor.execute("SELECT customer_id, name FROM customers")
customers = self.cursor.fetchall()
for customer in customers:
self.customer_id_combo.addItem(f"{customer[0]} - {customer[1]}", customer[0])
if current_customer_id == customer[0]:
self.customer_id_combo.setCurrentIndex(self.customer_id_combo.count() - 1)

def accept(self):
new_customer_id = self.customer_id_combo.currentData()
try:
new_room_no = int(self.room_no_input.text())
except ValueError:
QMessageBox.warning(self, "Error", "Room number must be an integer.")
return
try:
# Check if the new room number is already taken by another customer
self.cursor.execute(
"SELECT * FROM rooms WHERE room_no = %s AND customer_id != %s",
(new_room_no, self.room_id),
)
existing_room = self.cursor.fetchone()
if not existing_room:
# Update the room in the database
self.cursor.execute(
"UPDATE rooms SET customer_id = %s, room_no = %s WHERE room_id = %s",
(new_customer_id, new_room_no, self.room_id),
)
self.db.commit()
QMessageBox.information(self, "Success", "Room updated successfully!")
else:
QMessageBox.warning(self, "Error", "Room number already exists for another customer.")
except mysql.connector.Error as err:
QMessageBox.critical(self, "Error", f"Failed to update room: {err}")
super().accept()
```

COMPLETE PYTHON SOURCE CODE

```
class ManageBookDatesWindow(QWidget):
    def __init__(self, db, cursor, parent=None):
        super().__init__()
        self.db = db
        self.cursor = cursor
        self.setWindowTitle("Buenafe - Manage Booked Dates")
        self.setGeometry(100, 100, 800, 600)

        main_layout = QVBoxLayout()
        self.setLayout(main_layout)

        # Create table for booked dates
        self.booked_dates_table = QTableWidget()
        self.booked_dates_table.setColumnCount(3)
        self.booked_dates_table.setHorizontalHeaderLabels(
            ["Booking ID", "Customer Name", "Date Booked"]
        )
        self.load_booked_dates_data()
        main_layout.addWidget(self.booked_dates_table)

        # Create buttons for managing booked dates
        button_layout = QHBoxLayout()
        add_booked_date_button = QPushButton("Add Booked Date")
        add_booked_date_button.clicked.connect(self.add_booked_date)
        button_layout.addWidget(add_booked_date_button)

        edit_booked_date_button = QPushButton("Edit Booked Date")
        edit_booked_date_button.clicked.connect(self.edit_booked_date)
        button_layout.addWidget(edit_booked_date_button)

        delete_booked_date_button = QPushButton("Delete Booked Date")
        delete_booked_date_button.clicked.connect(self.delete_booked_date)
        button_layout.addWidget(delete_booked_date_button)

        main_layout.addLayout(button_layout)

    def load_booked_dates_data(self):
        # Clear the table
        self.booked_dates_table.setRowCount(0)

        try:
            # Fetch booked dates data (you'll need a bookings table)
            self.cursor.execute(
                "SELECT b.booking_id, c.name, b.date_booked "
                "FROM bookings b LEFT JOIN customers c ON b.customer_id = c.customer_id"
            )
            bookings = self.cursor.fetchall()
```

COMPLETE PYTHON SOURCE CODE

```
for booking in bookings:
row_position = self.booked_dates_table.rowCount()
self.booked_dates_table.insertRow(row_position)
for i, data in enumerate(booking):
item = QTableWidgetItem(str(data) if data is not None else "")
self.booked_dates_table.setItem(row_position, i, item)
except mysql.connector.Error as err:
QMessageBox.critical(self, "Error", f"Failed to load booked dates data: {err}")

def add_booked_date(self):
dialog = AddBookDateDialog(self, self.db)
if dialog.exec() == QDialog.DialogCode.Accepted:
customer_id, date_booked = dialog.get_booked_date_data()

# Input validation
if not customer_id or not date_booked:
QMessageBox.warning(self, "Error", "Please fill in all fields.")
return

try:
# Check if the booking date is already taken for the selected customer
self.cursor.execute("SELECT * FROM bookings WHERE customer_id = %s AND date_booked = %s", (customer_id,
date_booked))
if self.cursor.fetchone():
QMessageBox.warning(self, "Error", "The selected bookingdate is already taken.")
return

# Add booked date
self.cursor.execute("INSERT INTO bookings (customer_id, date_booked) VALUES (%s, %s)", (customer_id,
date_booked))
self.db.commit()

# Reload booked dates data
self.load_booked_dates_data()
QMessageBox.information(self, "Success", "Booked date added successfully.")
except mysql.connector.Error as err:
QMessageBox.critical(self, "Error", f"Failed to add booked date: {err}")

def edit_booked_date(self):
selected_row = self.booked_dates_table.currentRow()
if selected_row == -1:
QMessageBox.warning(self, "Error", "Please select a booked date to edit.")
return

booking_id = self.booked_dates_table.item(selected_row, 0).text()
customer_name = self.booked_dates_table.item(selected_row, 1).text()
date_booked = self.booked_dates_table.item(selected_row, 2).text()

dialog = EditBookDateDialog(self, self.db, booking_id, customer_name, date_booked)
if dialog.exec() == QDialog.DialogCode.Accepted:
customer_id, new_date_booked = dialog.get_booked_date_data()

# Input validation
if not customer_id or not new_date_booked:
QMessageBox.warning(self, "Error", "Please fill in all fields.")
return
```

COMPLETE PYTHON SOURCE CODE

```
try:
            # Check if the new booking date is already taken for the selected customer
            self.cursor.execute("SELECT * FROM bookings WHERE customer_id = %s AND
date_booked = %s", (customer_id, new_date_booked))
            if self.cursor.fetchone():
                QMessageBox.warning(self, "Error", "The selected booking date is already
taken.")
                return

            # Update booked date
            self.cursor.execute("UPDATE bookings SET customer_id = %s, date_booked = %s
WHERE booking_id = %s", (customer_id, new_date_booked, booking_id))
            self.db.commit()

            # Reload booked dates data
            self.load_booked_dates_data()
            QMessageBox.information(self, "Success", "Booked date updated successfully.")
        except mysql.connector.Error as err:
            QMessageBox.critical(self, "Error", f"Failed to update booked date: {err}")

    def delete_booked_date(self):
        selected_row = self.booked_dates_table.currentRow()
        if selected_row == -1:
            QMessageBox.warning(self, "Error", "Please select a booking to delete.")
            return

        booking_id_item = self.booked_dates_table.item(selected_row, 0)
        if not booking_id_item or not booking_id_item.text():
            QMessageBox.warning(self, "Error", "Invalid booking ID.")
            return

        booking_id = int(booking_id_item.text())
        customer_name = self.booked_dates_table.item(selected_row, 1).text()
        date_booked = self.booked_dates_table.item(selected_row, 2).text()

        confirm = QMessageBox.question(
            self,
            "Confirm Deletion",
            f"Are you sure you want to delete the booking for {customer_name} on
{date_booked}?",
            QMessageBox.StandardButton.Yes | QMessageBox.StandardButton.No,
            QMessageBox.StandardButton.No,
        )
```

COMPLETE PYTHON SOURCE CODE

```
if confirm == QMessageBox.StandardButton.Yes:
try:
self.cursor.execute("DELETE FROM bookings WHERE booking_id = %s", (booking_id,))
self.db.commit()
self.load_booked_dates_data()
QMessageBox.information(self, "Success", "Booking deleted successfully!")
except mysql.connector.Error as err:
QMessageBox.critical(self, "Error", f"Failed to delete booking: {err}")

def get_customer_id_by_name(self, name):
if not name: # Handle empty name
return None

try:
self.cursor.execute(
"SELECT customer_id FROM customers WHERE name = %s", (name,)
)

result = self.cursor.fetchone()
return result[0] if result else None # Extract the ID or return None
except mysql.connector.Error as err:
QMessageBox.critical(self, "Error", f"Database error: {err}")
return None

class AddBookDateDialog(QDialog):
def __init__(self, parent=None, db=None):
super().__init__(parent)
self.db = db
self.cursor = db.cursor()
self.setWindowTitle("Buenafe - Add Booked Date")

customer_id_layout = QHBoxLayout()
customer_id_layout.addWidget(QLabel("Customer ID:"))
self.customer_id_combo = QComboBox()
self.load_customer_ids()
customer_id_layout.addWidget(self.customer_id_combo)

date_booked_layout = QHBoxLayout()
date_booked_layout.addWidget(QLabel("Date Booked:"))
self.date_booked_edit = QDateEdit()
date_booked_layout.addWidget(self.date_booked_edit)

layout = QVBoxLayout()
layout.addLayout(customer_id_layout)
layout.addLayout(date_booked_layout)

button_box = QDialogButtonBox(QDialogButtonBox.StandardButton.Ok | QDialogButtonBox.StandardButton.Cancel)
button_box.accepted.connect(self.accept)
button_box.rejected.connect(self.reject)
layout.addWidget(button_box)

self.setLayout(layout)

def load_customer_ids(self):
self.cursor.execute("SELECT customer_id, name FROM customers")
customers = self.cursor.fetchall()
for customer in customers:
self.customer_id_combo.addItem(f"{customer[0]} - {customer[1]}", customer[0])
```

COMPLETE PYTHON SOURCE CODE

```
def get_booked_date_data(self):
        customer_id = self.customer_id_combo.currentData()
        date_booked = self.date_booked_edit.date().toPyDate()  # Get the selected date
        return customer_id, date_booked

class EditBookDateDialog(QDialog):
    def __init__(self, parent=None, db=None, booking_id=None, customer_id=None, date_booked=None):
        super().__init__(parent)
        self.db = db
        self.cursor = db.cursor()
        self.booking_id = booking_id
        self.customer_id = customer_id
        self.date_booked = date_booked
        self.setWindowTitle("Buenafe - Edit Booked Date")

        customer_id_layout = QHBoxLayout()
        customer_id_layout.addWidget(QLabel("Customer ID:"))
        self.customer_id_combo = QComboBox()
        self.load_customer_ids(customer_id)
        customer_id_layout.addWidget(self.customer_id_combo)

        date_booked_layout = QHBoxLayout()
        date_booked_layout.addWidget(QLabel("Date Booked:"))
        self.date_booked_edit = QDateEdit()
        date_text = QDate.fromString(date_booked, "yyyy-MM-dd")
        self.date_booked_edit.setDate(date_text)
        date_booked_layout.addWidget(self.date_booked_edit)

        layout = QVBoxLayout()
        layout.addLayout(customer_id_layout)
        layout.addLayout(date_booked_layout)

        button_box = QDialogButtonBox(QDialogButtonBox.StandardButton.Ok | QDialogButtonBox.StandardButton.Cancel)
        button_box.accepted.connect(self.accept)
        button_box.rejected.connect(self.reject)
        layout.addWidget(button_box)

        self.setLayout(layout)

    def load_customer_ids(self, customer_id):
        query = "SELECT customer_id, name FROM customers"
        self.cursor.execute(query)
        customers = self.cursor.fetchall()
        for customer in customers:
            self.customer_id_combo.addItem(f"{customer[0]} - {customer[1]}", customer[0])
        self.customer_id_combo.setCurrentText(str(customer_id))

    def get_booked_date_data(self):
        customer_id = self.customer_id_combo.currentData()
        date_booked = self.date_booked_edit.date().toPyDate()
        return customer_id, date_booked

    def accept(self):
        new_customer_id, new_date_booked = self.get_booked_date_data()

        try:
            # Check if the new booking date is already taken for the selected customer
            self.cursor.execute(
                "SELECT * FROM bookings WHERE customer_id = %s AND date_booked = %s AND booking_id != %s",
                (new_customer_id, new_date_booked, self.booking_id),
            )
            existing_booking = self.cursor.fetchone()
```

COMPLETE PYTHON SOURCE CODE

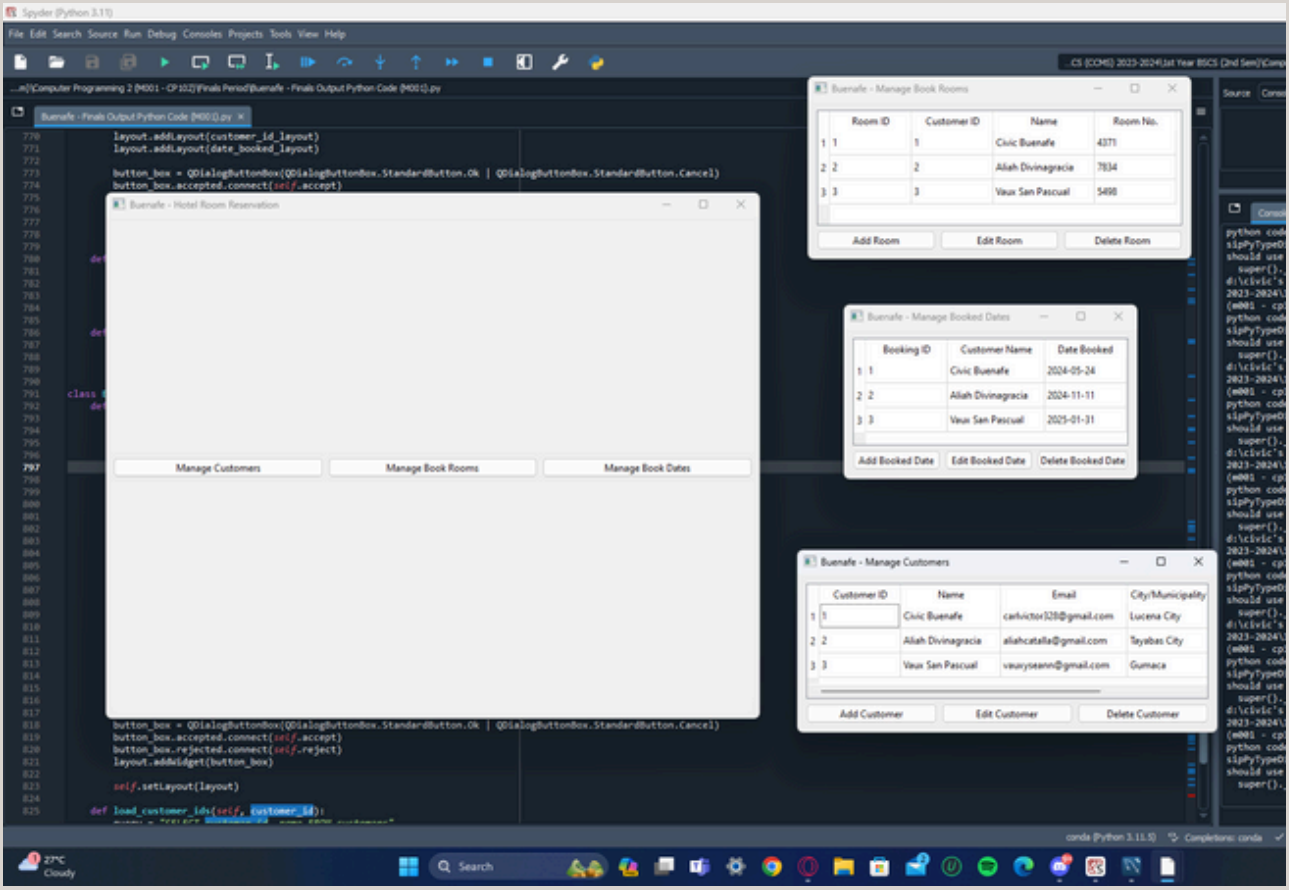
```
if not existing_booking:
# Update the booked date in the database
self.cursor.execute(
"UPDATE bookings SET customer_id = %s, date_booked = %s WHERE booking_id = %s",
(new_customer_id, new_date_booked, self.booking_id),
)
self.db.commit()
QMessageBox.information(self, "Success", "Booked date updated successfully!")
else:
QMessageBox.warning(self, "Warning", "This booking date is already taken for the selected customer!")
except mysql.connector.Error as error:
QMessageBox.critical(self, "Error", str(error))
finally:
self.db.commit()
self.close()

# Open the dialog
dialog = EditBookedDateDialog(1, 1, QDate.fromString("2022-01-01", "yyyy-MM-dd"), db)
if dialog.exec():
print("Booked date updated successfully!")
else:
print("No changes made.")


if __name__ == "__main__":
db_config = {
'host': 'localhost',
'user': 'root',
'password': 'civic328',
'database': 'hoteldatabase'
}

app = QApplication(sys.argv)
main_window = MainWindow()
main_window.show()
sys.exit(app.exec())
```

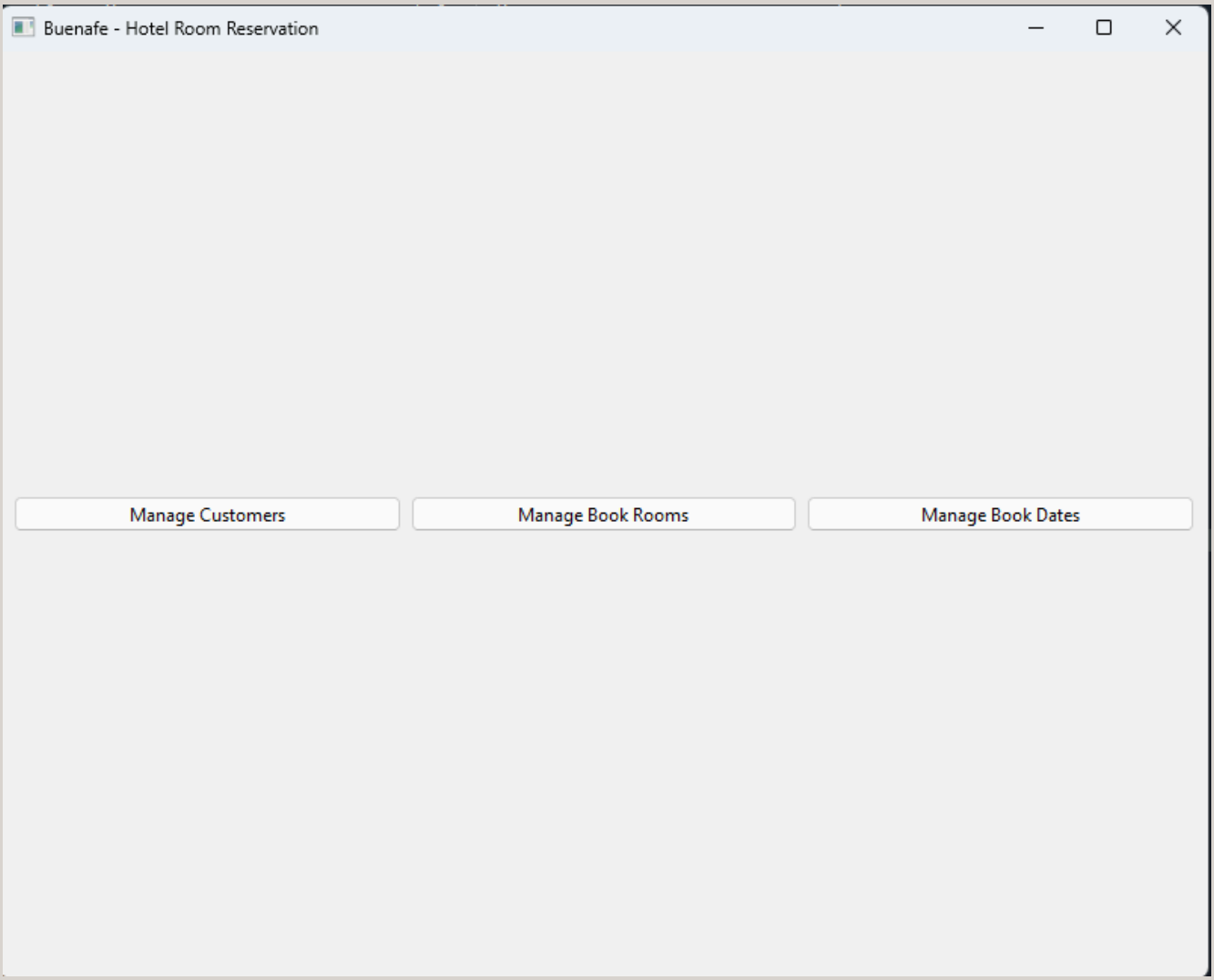
SCREENSHOTS OF OUTPUT



Total of 4 Windows including the Main Window and it has a other 3 windows inside the Main Window are “Manage Customer”, “Manage Book Rooms”, and “Manage Book Dates

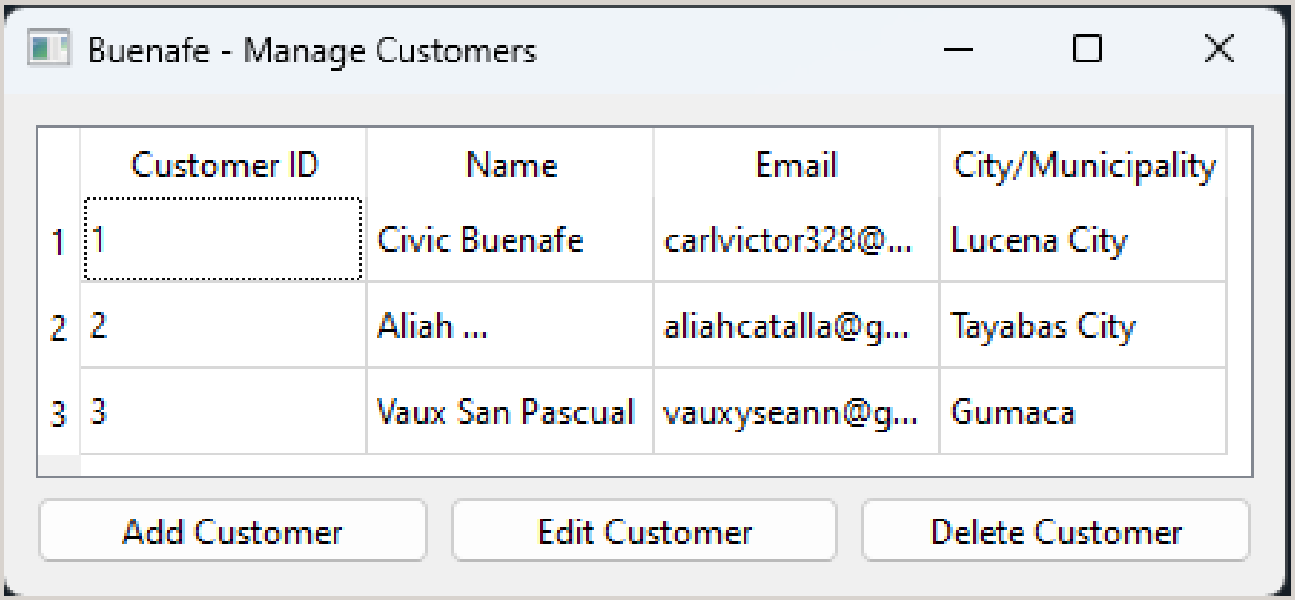
SCREENSHOTS OF OUTPUT

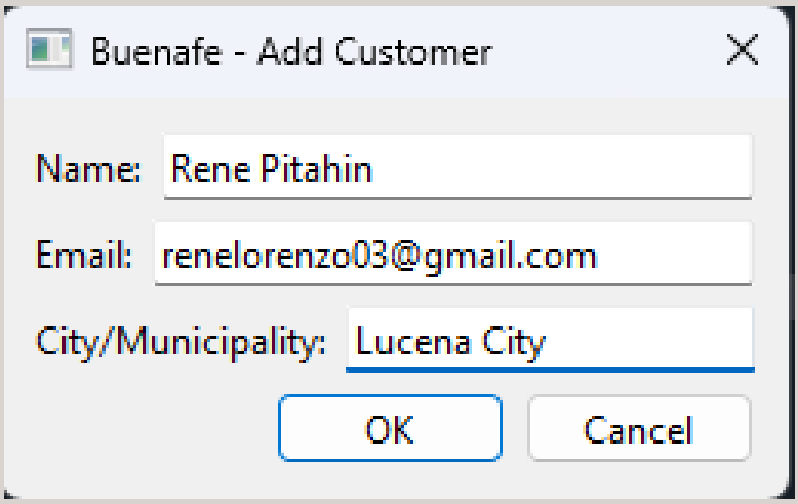
Main Window

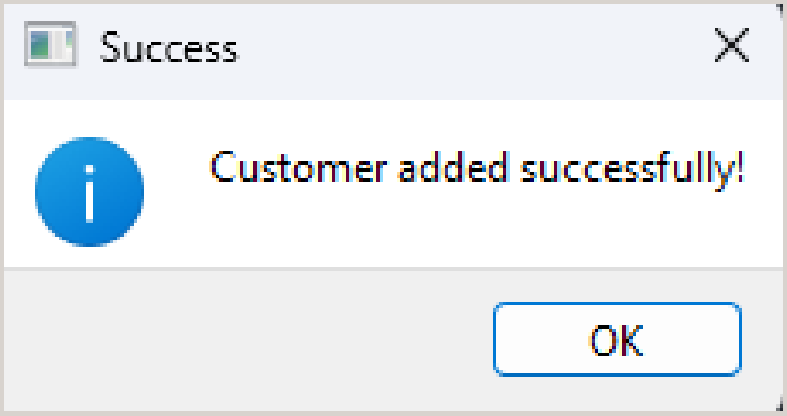


SCREENSHOTS OF OUTPUT

Manage Customer Window

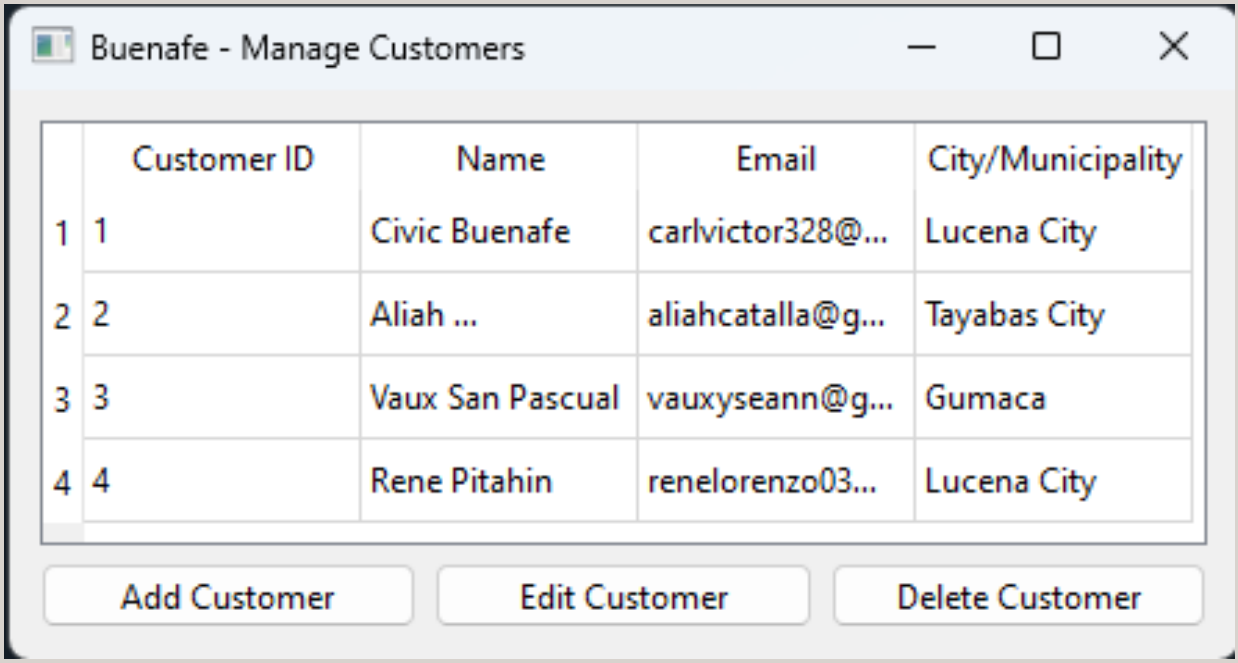


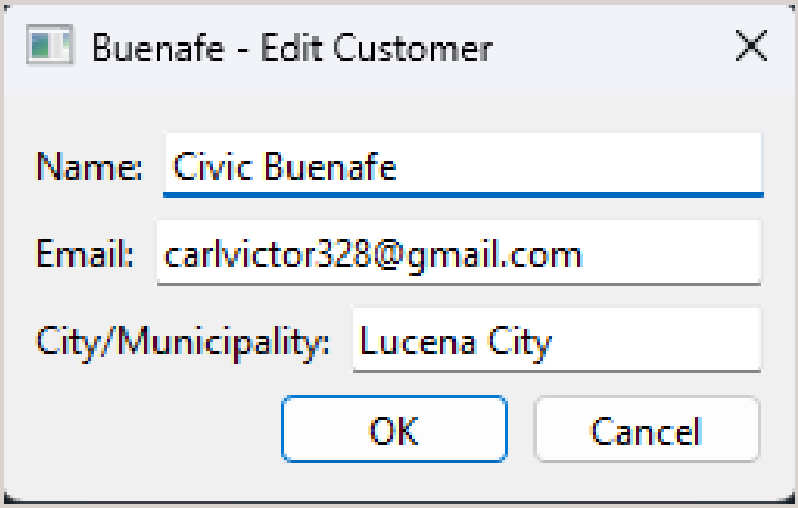


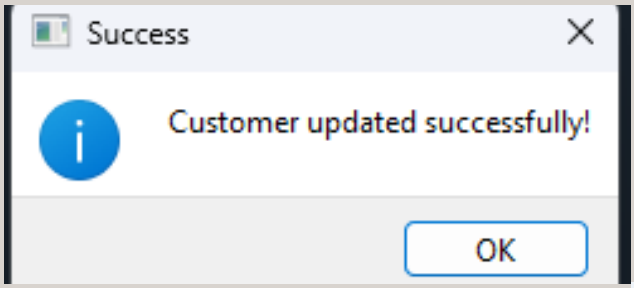


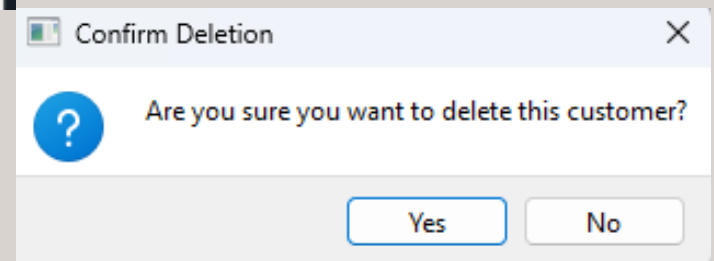
SCREENSHOTS OF OUTPUT

Manage Customer Window



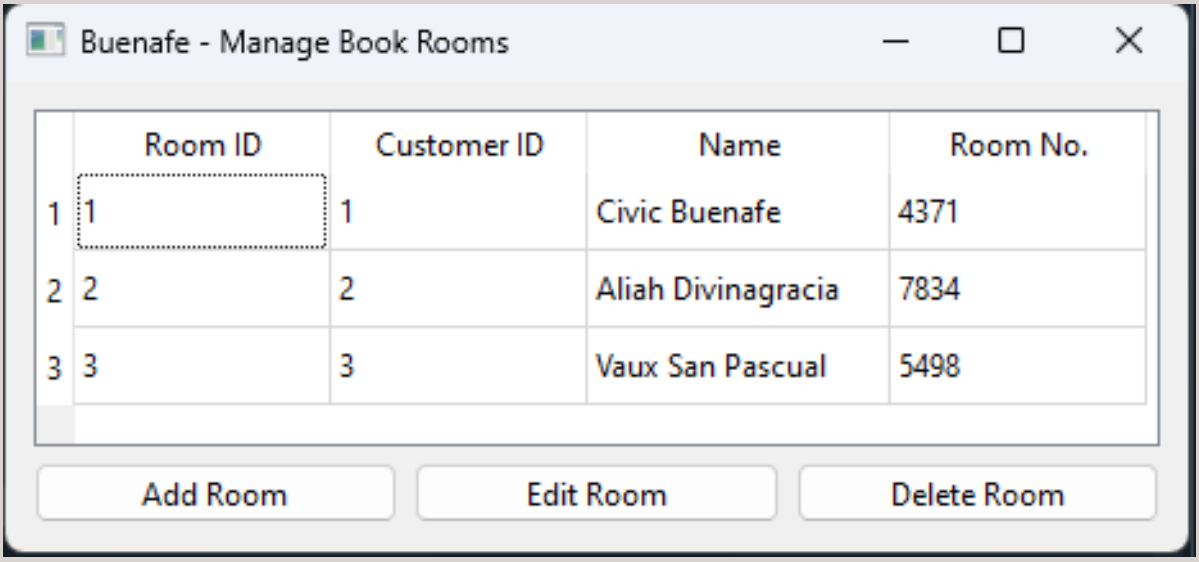


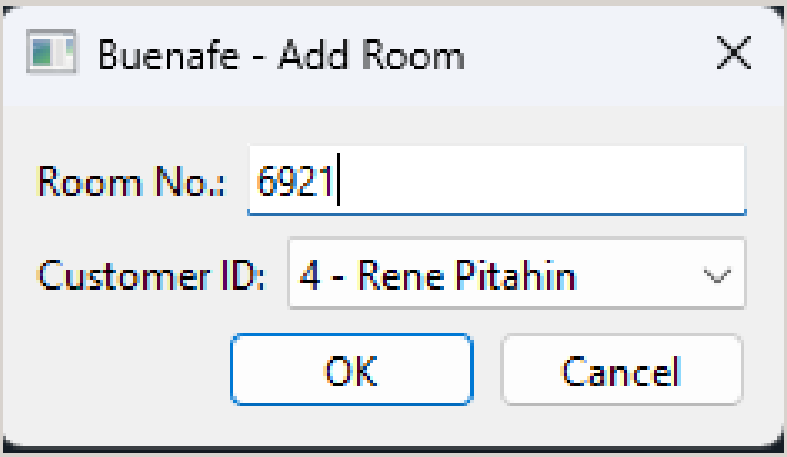


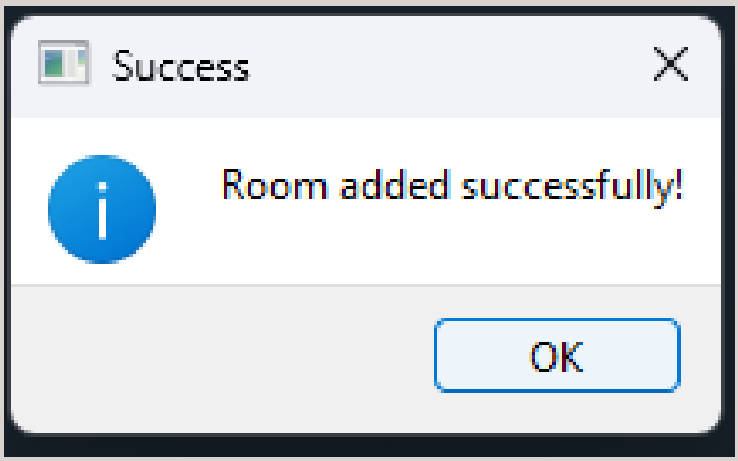


SCREENSHOTS OF OUTPUT

Manage Book Rooms Window

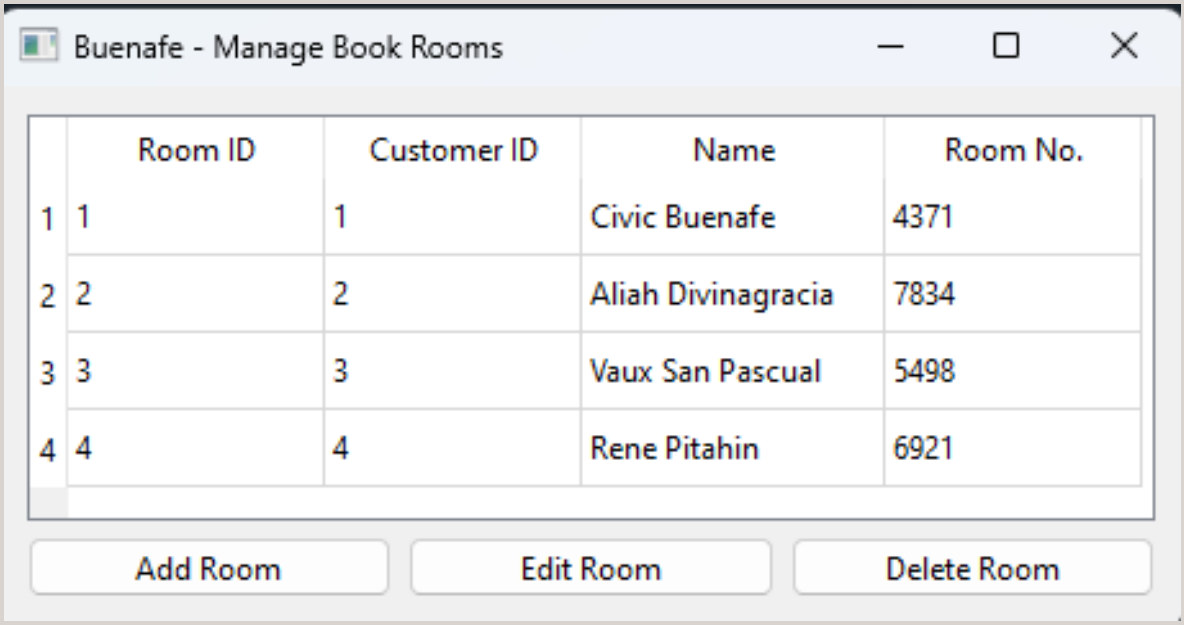


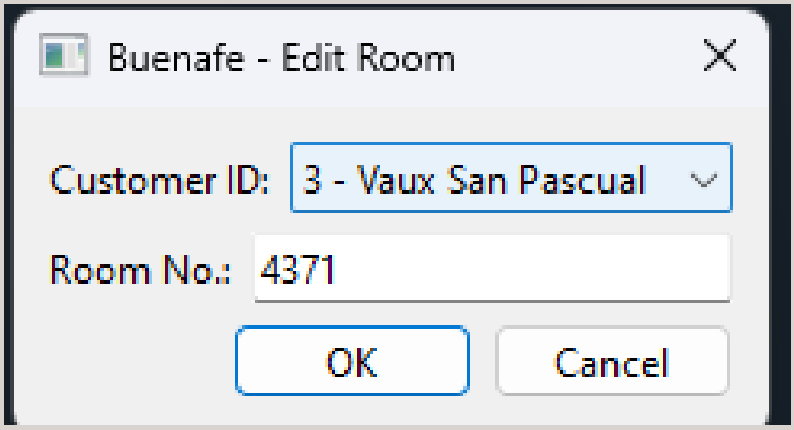


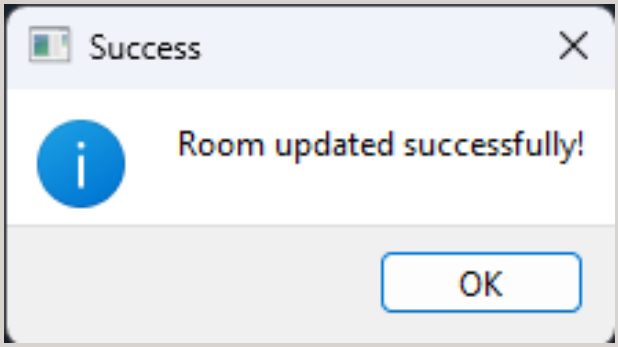


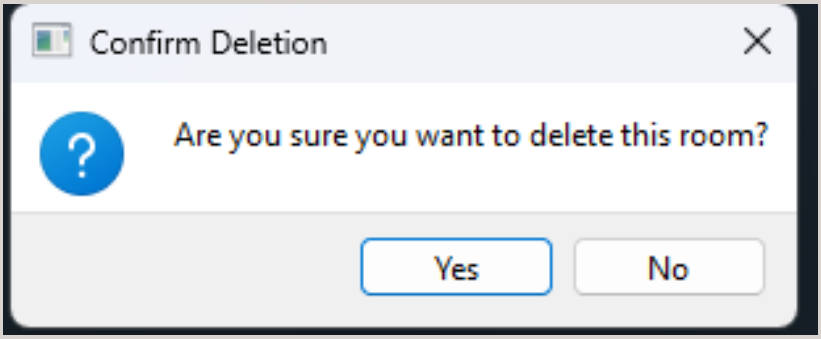
SCREENSHOTS OF OUTPUT

Manage Book Rooms Window



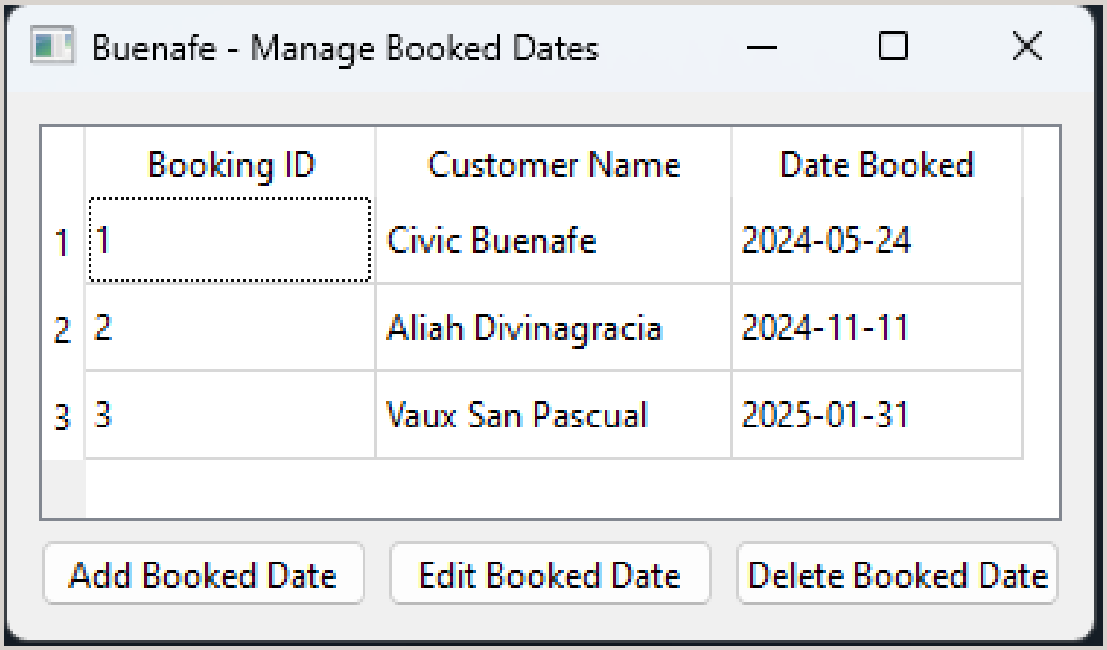


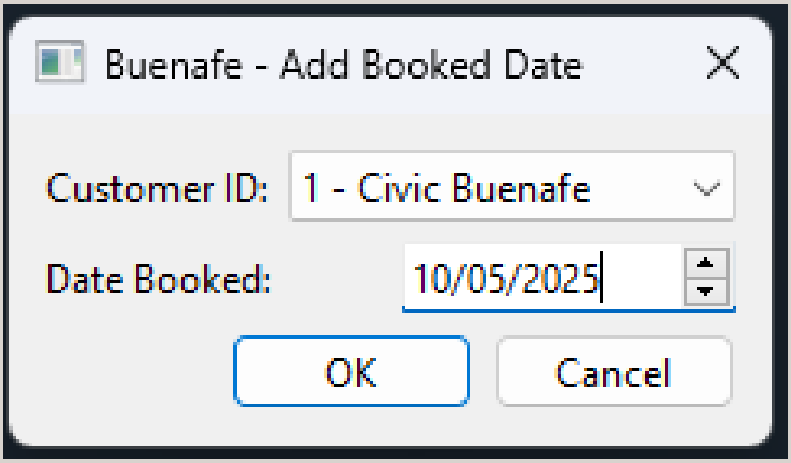


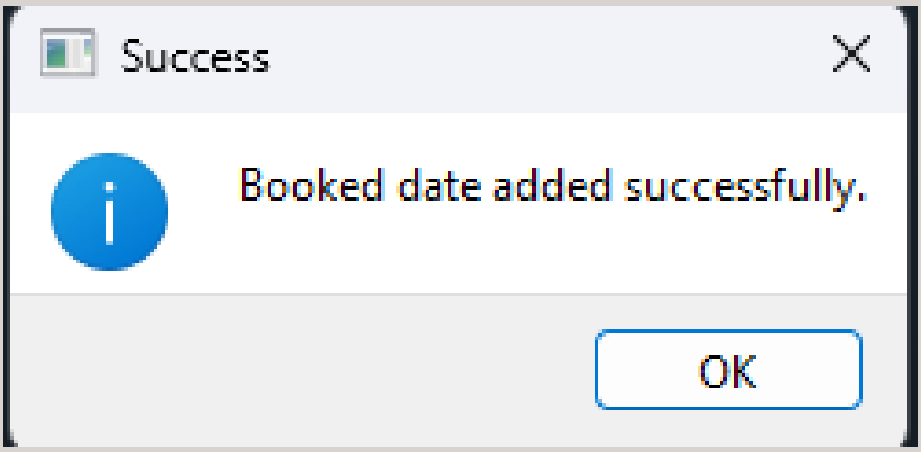


SCREENSHOTS OF OUTPUT

Manage Book Dates Window

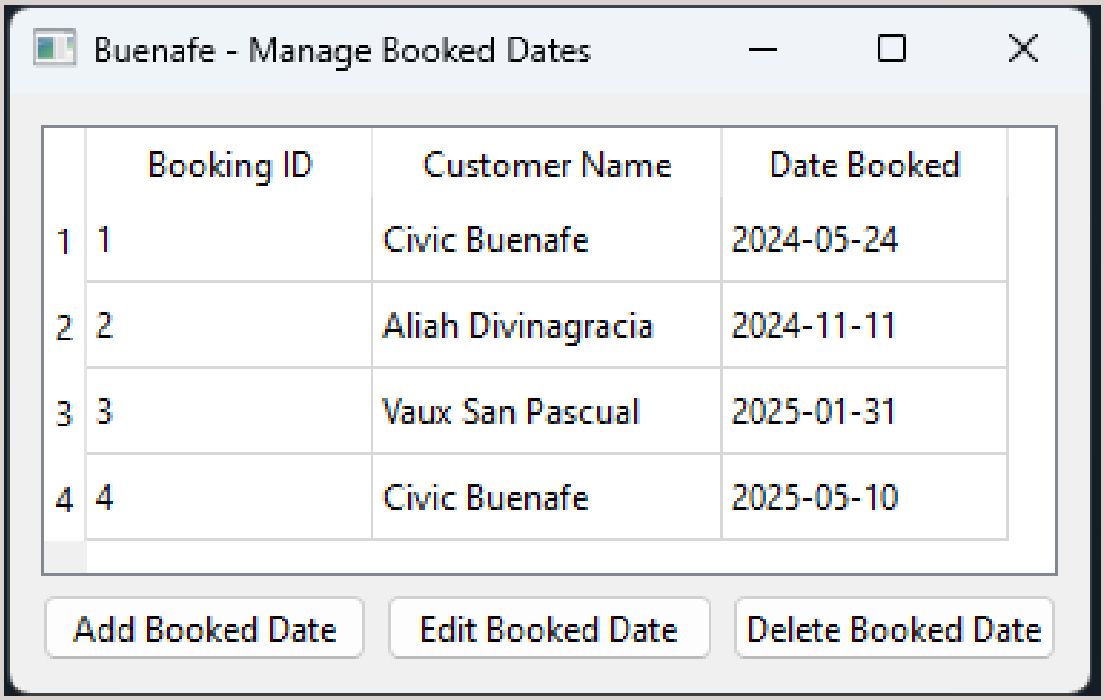


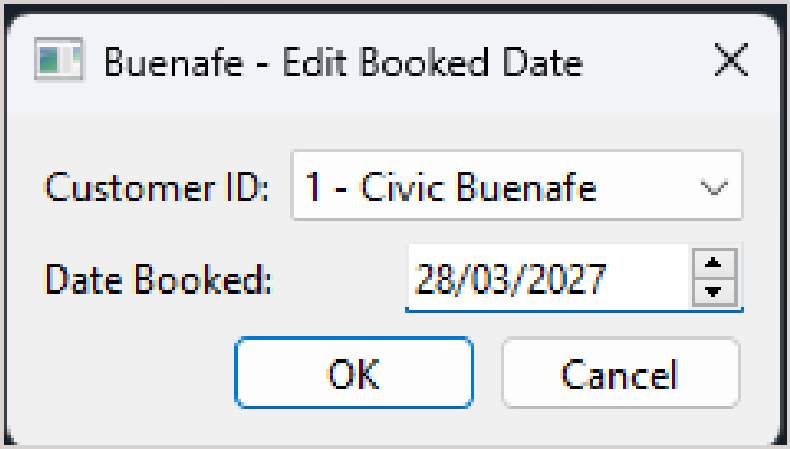


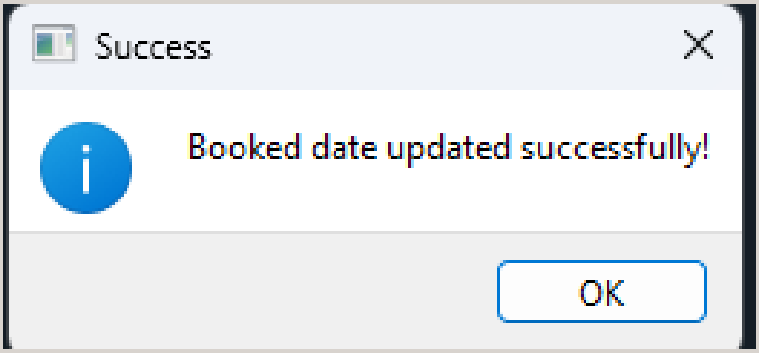


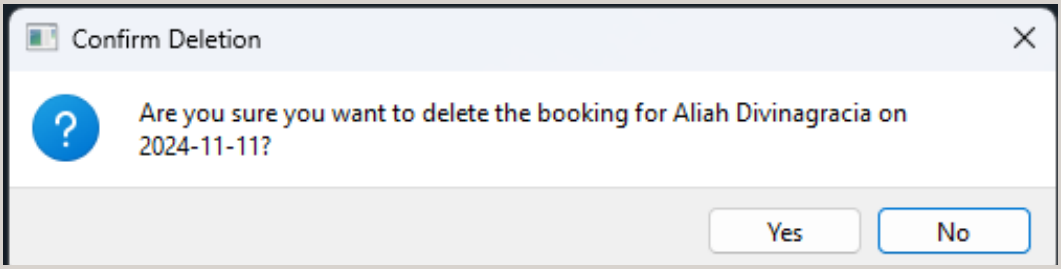
SCREENSHOTS OF OUTPUT

Manage Book Dates Window









LEARNING REFLECTION

IN COMPUTER PROGRAMMING 2, I EXPLORED THE LEARNING OBJECTIVES RELATED TO SQL, PYTHON, AND PYQT6. THIS FINAL PERIOD IS BEEN ONE OF THE THOUGHEST PART IN MY COLLEGE JOURNEY ESPECIALLY IN THIS MAJOR SUBJECT. I LEANRED FROM MYSELF THAT HOW TO DEBUG A ERROR CODE WHEN IT IS NOT WORKING, WHEN IT NOT CONENCTING TO SQL AND PYQT6 DESIGNER. I LEARNED ALSO HOW TO CREATE A GUI DESIGN BY CREATING YOUR OWN WINDOW DESIGN.

WORKING ON THIS FINAL COURSE OUTPUT PROJECT, IT BECAME ONE OF THE HARDEST FOR ME TO DEBUG A CODE. IT BECAME A THOUGH DAY FOR ME FOR CREATING A CODE FOR 2 DAYS, THE MORE THAT I HAVING A ERROR, THE MORE THAT I UNDERSTAND THE REASON WHY IT HAS ERRORS, BECAUSE WE CAN LEARN IT ON HOW TO SOLVE THESE PROBLEMS EXAMPLE ARE CONNECTING PYTHON AND SQL, PYTHON AND PYQT6.ONE CHALLENGE WAS DISPLAYING DATABASE INFORMATION FOR EDITING. I TRIED QTABLEWIDGETS BUT COULDN'T GET THEM TO WORK. IT TOOK SOME TIME AND EXPERIMENTATION, BUT EVENTUALLY, I SUCCESSFULLY USED QLINEEDITS AND A QCOMBOBOX TO SHOW AND MODIFY DATA.

I ENCOUNTERED MINOR ISSUES WITH SQL AND INPUT RESTRICTIONS, WHICH I RESOLVED WITH SMALL FIXES. THIS PROJECT GAVE ME VALUABLE EXPERIENCE PROGRAMMING ACROSS MULTIPLE PLATFORMS, A SKILL I KNOW WILL BE ESSENTIAL IN THE FUTURE.