HOTEL LODGE
MANAGEMENT SYSTEM

Report

PROJECT
REPORT

NAME : YASH PRATAP SINGH
REG NO. : 25MIM10193
COURSE :PYTHON ESSENTIALS
FACULTY: DR MURUGESWARI K

PYTHON ESSENTIALS
Slot: B11+B12+B13

INTRODUCTION

The Hotel Management System is a modular Python application designed to simplify hotel operations such as room booking, guest management, billing, and checkout. Built using Tkinter, the system offers a simple and intuitive desktop interface for small lodges and hotels. The backend is structured using multiple Python modules to ensure clean architecture, maintainability, and adherence to academic project standards.
This project follows VITyarthi Build Your Own Project guidelines and includes functional modules, non-functional requirements, documentation, testing, and design structure.

Hotel Management System simplifies hotel operations by managing guest bookings, room availability, billing, and checkouts. It automates daily tasks, reduces errors, and provides an organized way to record customer details. With a user-friendly interface, it helps staff efficiently handle reservations, track occupancy, and improve overall service quality.

PROBLEM STATEMENT

Small hotels and lodges require a simple and efficient way to:

- Register guests and book room.
- Track room availability by room type
- Generate bills based on room type and number of nights
- Checkout guests and free rooms

Manual processes can result in double bookings, miscalculated bills, and difficulty in tracking room occupancy. The aim is to replace such manual methods with an easy-to-use software solution that runs on a single computer at the hotel front desk.

FUNCTIONAL REQUIREMENTS

The system provides the following functional modules:

1. Guest Registration & Booking

- Input guest name and phone number.
- Select room type (Single, Double, Suite).
- Enter number of nights.
- System automatically sets check-in date (current date) and computes check-out date.
- Validate input and show error messages for invalid entries.

2. Room Inventory Management

- Maintain available room counts:
- Single: 5 rooms
- Double: 3 rooms
- Suite: 2 rooms
- On booking, decrease the corresponding room count.
- Prevent booking when no rooms are available for the chosen type.
- On checkout, increase the room count back.

3. Guest Listing

- Display all active guests in a table with:
- Name
- Phone
- Room Type
- Nights
- Check-In Date
- Check-Out Date

4. BILLING

- When a guest is selected from the table, display bill details:
- Room type
- Nights stayed
- Rate per night
- Total amount

5.CHECKOUT

- Allow staff to checkout a selected guest.
- Remove the guest from the active guest list.
- Increase the available room count for that room type.

On-functional Requirements

1. Usability
- Intuitive GUI with clearly labeled fields and buttons.
- User prompts and error messages guide the receptionist.

2. Reliability
- Validations prevent invalid data from being stored.
- Room inventory is kept consistent during bookings and checkouts.

3. Maintainability
- The code is modular: separate files for models, storage, validation, booking logic, billing, and UI.
- Functions have clear responsibilities and are documented with comments.

4. Performance
- All data is stored in memory; operations like booking and listing are instantaneous for typical small hotel usage.

5. Error Handling
- User-friendly error dialogs for invalid phone numbers, invalid nights, and no guest selected for checkout.
- Catch-all handler in the GUI to log/display unexpected errors.

System Architecture

The system uses a layered architecture:

Presentation Layer (UI)

- - Tkinter GUI (`ui.py`): handles user input and displays data.

Business Logic Layer

- - Booking logic (`booking.py`): validates data, checks inventory, creates bookings.
- - Billing logic (`billing.py`): computes the bill based on room rates and nights.

Data Layer

- - Models (`models.py`): defines the `Guest` data class and room configuration.
- - Storage (`storage.py`): manages in-memory lists for guests and room counts.

High-level architecture diagram (textual):

- User → Tkinter GUI (ui.py) → Booking/Billing Logic (booking.py, billing.py) → Storage (storage.py) → Models (models.py)

Design Diagram

7.1 Use Case Diagram (Textual Description)

Actors:
- Receptionist

Use Cases:
- UC1: Register Guest and Book Room
- UC2: View Current Guests
- UC3: View Bill for Guest
- UC4: Checkout Guest

R**elationships:**
- Receptionist performs all use cases.
- UC1 includes validation of inputs.
- UC4 includes bill viewing (optional) before checkout.

7.2 Workflow Diagram (Booking)

1. Receptionist opens application.
2. Enters guest name and phone number.
3. Selects room type.
4. Enters number of nights.
5. Clicks **Book Room**.
6. System validates inputs.
7. System checks room availability.
8. If valid and available:
- Creates booking
- Updates room inventory
- Shows success message
9. If invalid:
- Shows appropriate error message.

7.3 Sequence Diagram (Guest Booking – Textual)

1. **Receptionist→ UI:** Enter details and click "Book Room".
2. **UI** → **Booking Logic**: book_guest(name, phone, room_type, nights_str)
3. **Booking Logic** →Validation: is_valid_phone(phone)`, `parse_nights(nights_str)
4.**Booking Logic→ Storage:**get_available_rooms(room_type)
5. **Booking Logic → Storage:** allocate_room(room_type)` and `add_guest(guest)
6. **Booking Logic → UI**: Return success/guest object.
7. **UI → Receptionist**: Show confirmation message and update table.

7.4 Class Diagram (Simplified – Textual)

Class: Guest
- Attributes: name, phone, room_type, nights, check_in, check_out

Module: storage
- Data: `_guests: List[Guest]`, `_rooms: Dict[str, int]`
- Functions:
- `get_guests()`
- `add_guest(guest)`
- `remove_guest(index)`
- `get_available_rooms(room_type)`
- `allocate_room(room_type)`
- `release_room(room_type)`

Module: booking
- Functions:
- `book_guest(name, phone, room_type, nights_str)`
- `checkout_guest(index)`

Module: billing
- Functions:
- `calculate_bill(guest)`
- `format_bill_text(guest)`

Module: validation
- Functions:
- `is_valid_phone(phone)`
- `parse_nights(nights_str)`

Module: ui
- Functions:
- `run_app()`
- `on_book_room()`
- `on_checkout_guest()`
- `on_show_bill()`
- `refresh_table()`

7.5 ER Diagram (Logical)

Entities:
1. **Guest**
- GuestName
- Phone
- RoomType
- Nights
- CheckInDate
- CheckOutDate

2**. RoomType**
- RoomTypeName
- AvailableCount
- RatePerNight

Relationship:
- One **RoomType** can be associated with many **Guest** bookings (1-to-many).

Currently, the implementation uses **in-memory data structures** instead of a physical database.

Design Decisions & Rationale

1. Tkinter for GUI
- Tkinter is part of the Python standard library, so no external dependencies are needed.

2. In-Memory Storage
- For a small demo project and academic purpose, using lists and dictionaries is simple and sufficient.

3. Modular Architecture
- Separating UI, business logic, and storage improves maintainability and readability.
- Easier to extend or replace components in the future (e.g., replacing storage with a database).

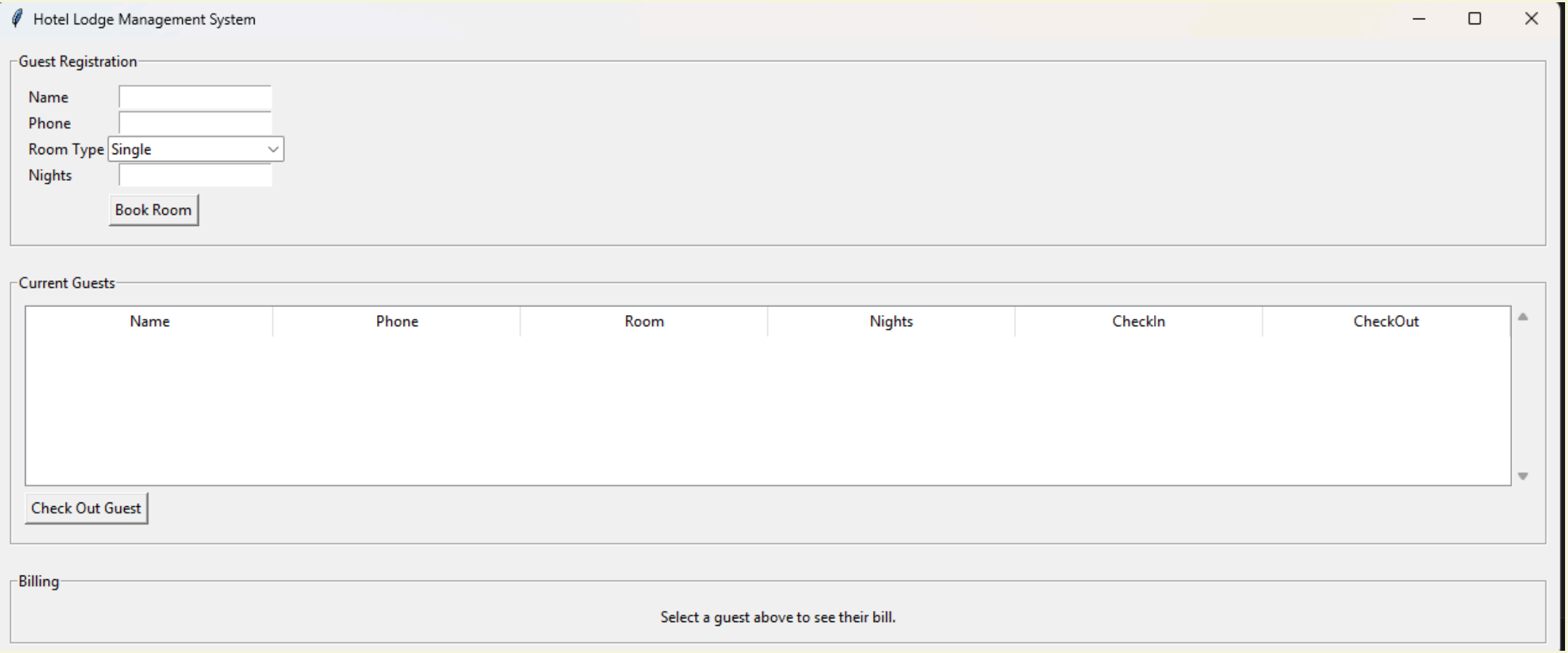
4. Validation and Error Handling
- Centralized validation functions reduce duplication.
- Custom exceptions (`ValidationError`, `BookingError`) make it easier to handle errors in the UI layer.

Implementation Details

Language: Python 3.x
Main Files:
- `src/main.py` – Application entry point
- `hotel_management/models.py` – Guest model and configuration
- `hotel_management/storage.py` – In-memory storage
- `hotel_management/validation.py` – Input validation
- `hotel_management/booking.py` – Booking and checkout logic
- `hotel_management/billing.py` – Bill calculation
- `hotel_management/ui.py` – Tkinter GUI implementation

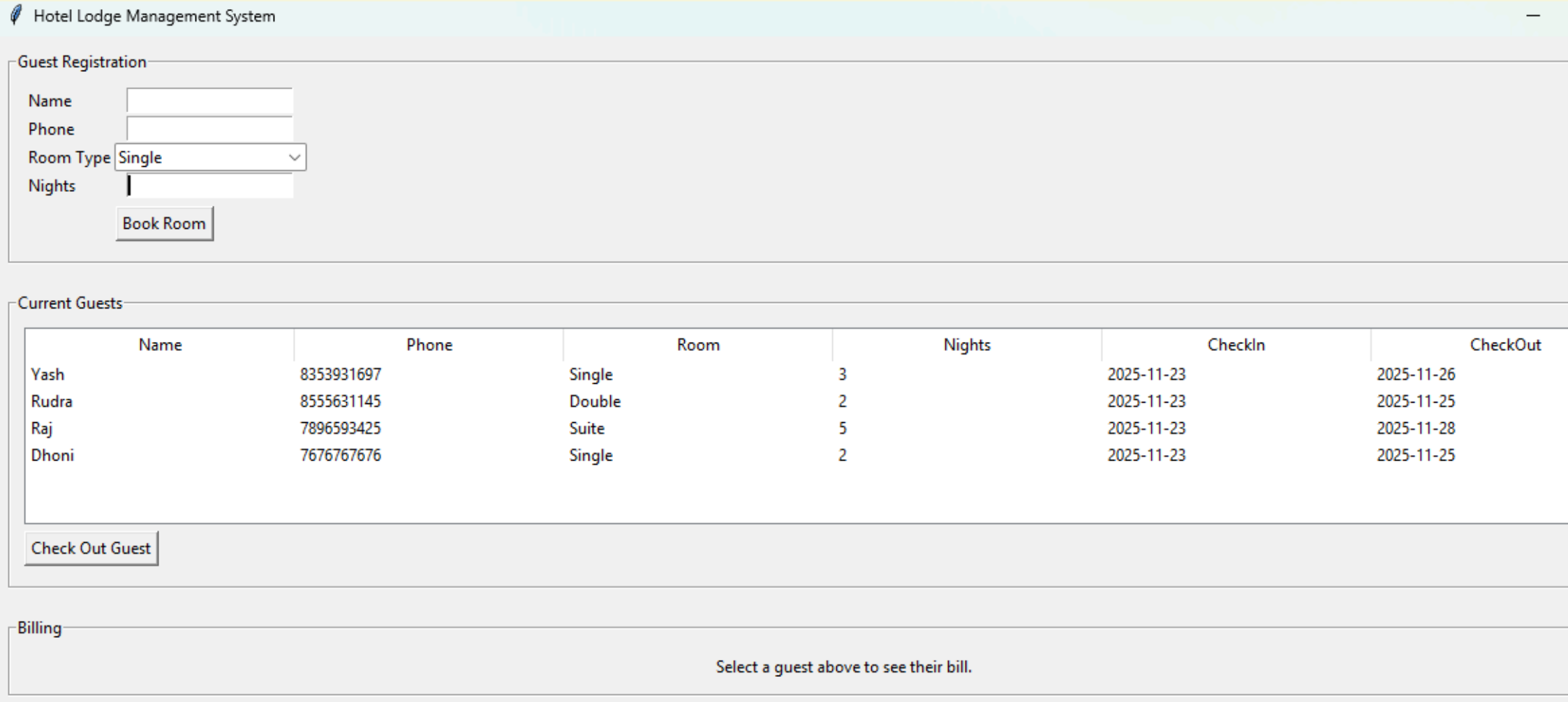
Implementation steps included:
1. Starting with a single-file Tkinter program.
2. Refactoring into multiple modules to satisfy project requirements.
3. Adding clearer validation and exception handling.
4. Introducing a `Guest` data class for structured data storage.
5. Adding basic unit tests for validation logic.

1. Application main window.



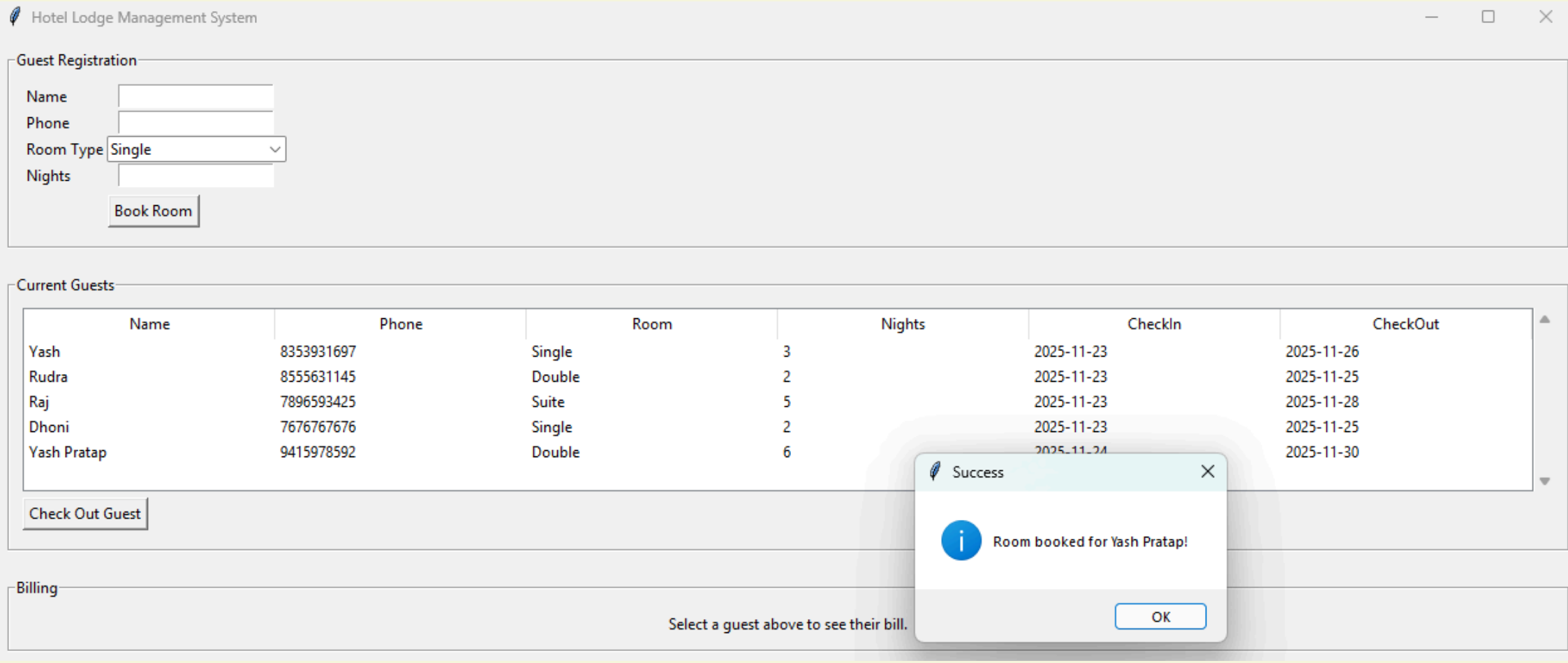
Please zoom in to get a clearer view of the image.

2. Screen showing multiple guests in the table.



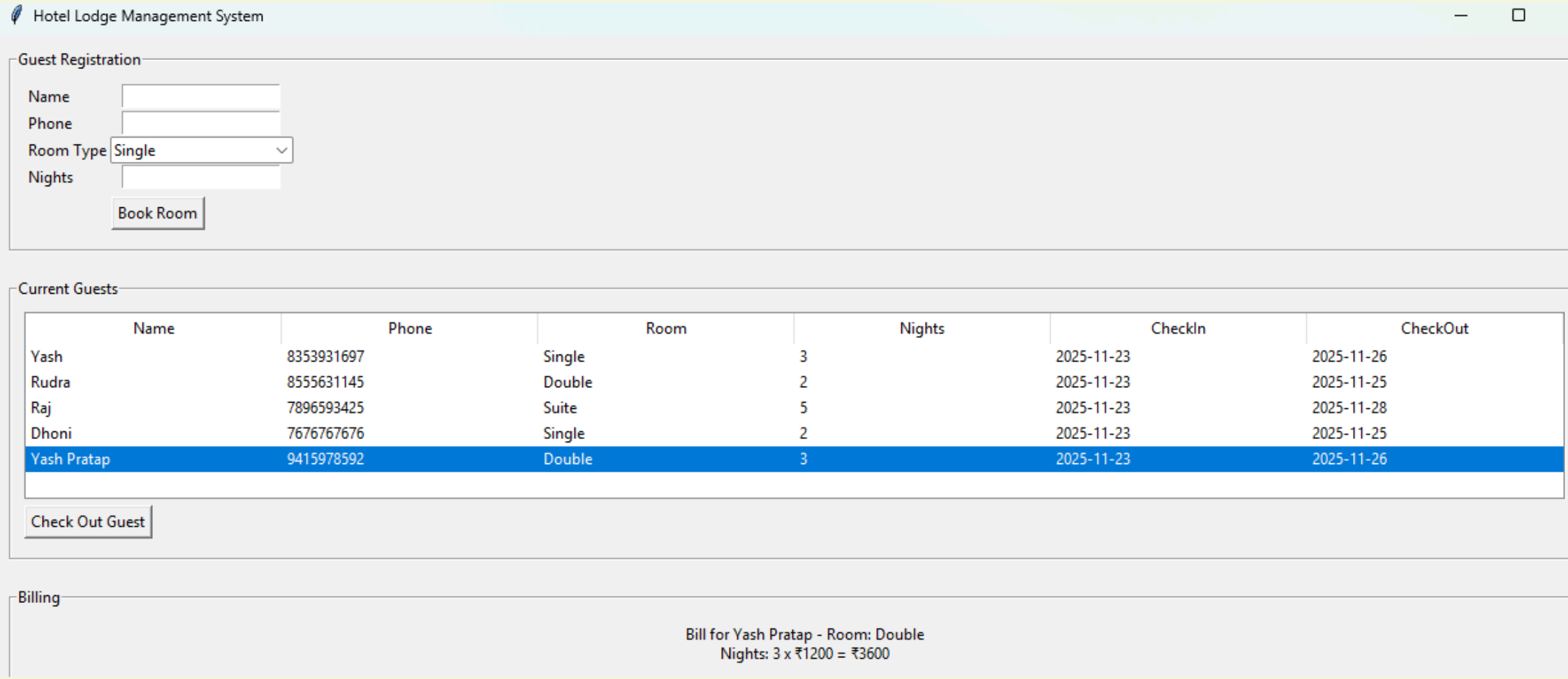
Please zoom in to get a clearer view of the image.

3.Confirmation dialog for a successful booking.



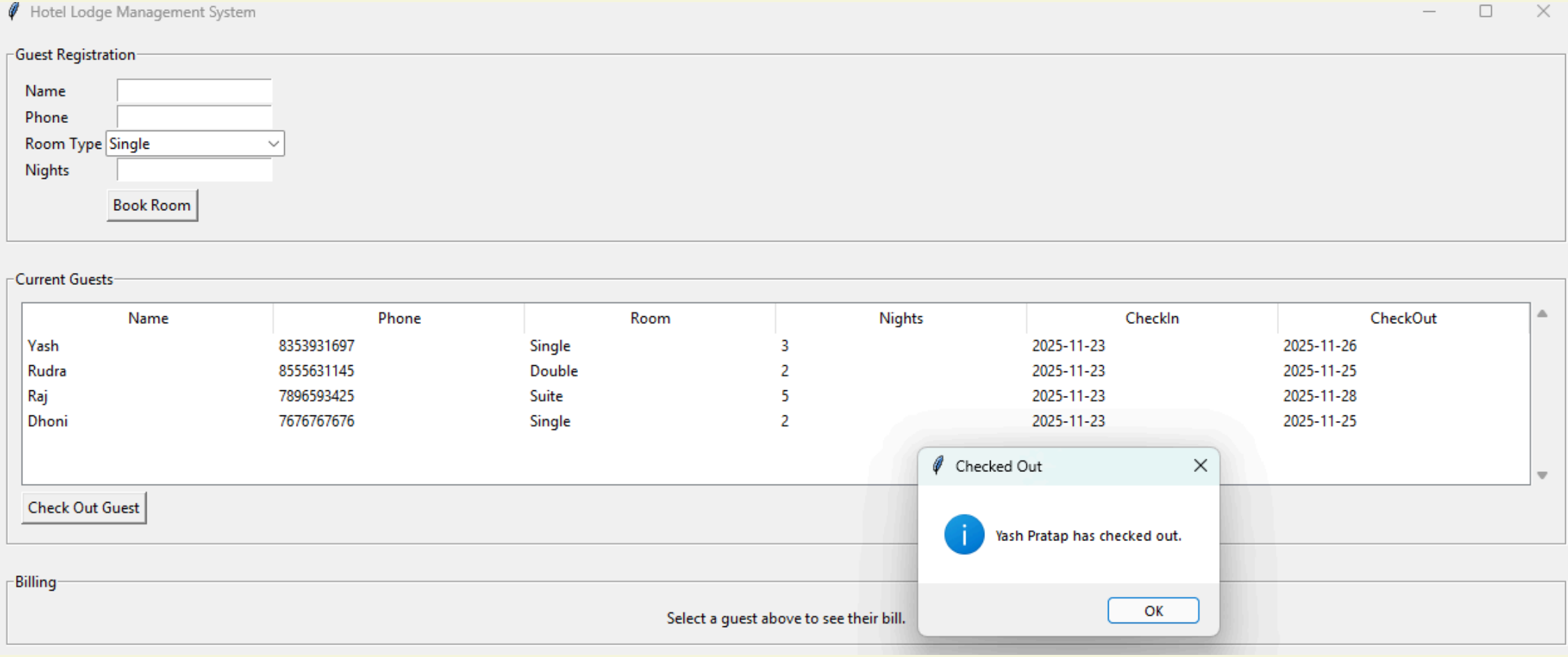
Please zoom in to get a clearer view of the image.

4.Billing information shown for a selected guest.



Please zoom in to get a clearer view of the image.

5. Confirmation dialog for successful checkout.



Please zoom in to get a clearer view of the image.

Testing Approach

Type of Testing:

- Manual testing of GUI interactions.
- Automated unit tests for validation functions.

Unit Tests:

- Valid and invalid phone numbers.
- Valid and invalid nights inputs (empty, non-numeric, zero).

Manual Test Scenarios:

1. Book a Single room for valid guest details.
2. Try booking with invalid phone number (less than 10 digits).
3. Try booking with empty nights.
4. Try booking when room inventory is exhausted for a specific type.
5. Select a guest and verify bill calculation.
6. Checkout a guest and verify they are removed and availability increases.

Challenges Faced

- Designing a clean and modular structure from an initial single-file implementation.
- Ensuring that room inventory remains consistent under all operations.
- Managing GUI events and updating the table correctly after each operation.
- Handling user errors gracefully in the GUI without crashing the application.

Learnings & Key Takeaways

- How to design and implement a GUI application using Tkinter.
- Importance of modular programming and separating concerns (UI, logic, data).
- How input validation and error handling improve reliability and user experience.
- Basics of modeling real-world entities (guests, rooms) using Python data classes.
- Experience with writing and running unit tests for critical functions.

Future Enhancements

- Add persistent storage using SQLite or another database.
- Implement user authentication (admin/receptionist login).
- Add filters and search functionality in the guest table.
- Generate PDF bills or invoices for each guest.
- Implement reporting features such as:
  - Daily revenue report
  - Occupancy percentage
  - Historical booking trends

References

- Python official documentation (Tkinter, datetime, dataclasses)
- Course lecture notes and lab materials
- Online resources for Python GUI examples and best practices

THANK YOU