Hotel Havana Management System - Project Report

Cover Page

Hotel Havana Management System
VITYARTHI Project
Developed using Python, Tkinter, and tkcalendar
November 2025

Introduction

Finding hotels irritating? Many systems either too heavy or too clunky. Wanted something easier—simple, but working well. So we built Hotel Havana. You can book rooms fast, handle records, and track payments. All on a nice GUI. Short, sharp, solid.

Problem Statement

Hotels need to handle room bookings, track payments, and search records. Most tools? Too big or too slow. Smaller hotels want fast, direct solutions. Must work on just Python—no big database server.

Functional Requirements

- Book rooms by providing guest details and dates.
- System should auto-assign room number and customer ID.
- Records can be searched by name or phone number.
- Payment status tracked within the app.
- App must offer calendar date-entry widgets.
- Must run fully offline. Local execution only.
- Show error for duplicate phone numbers with pending payment.

Non-functional Requirements

- Runs only on Python 3.13.7 or above.
- Tkinter must be available.
- tkcalendar required for date picking.
- Response time—under 2 seconds for every main app operation.
- Interface must look clean and be simple to navigate.

- Data stays only in memory—no file or db usage.
- One window; not multiple pop-ups all over the place.

System Architecture

Creates one main application window using Tkinter. All features—booking, records, payment—sit as menus in this single window. Data? Just Python lists. No saving to disk. Refreshes interface when needed by destroying/rebuilding widgets.

Block Diagram (Textual):

- Main Window
    - Booking Menu
    - Records Menu
    - Payment Menu

All access same in-memory data lists.

Design Diagrams

Hand-drawn or tool-generated diagrams can be inserted in delivered Google Doc.

Use Case Diagram

Actors:

- **Customer**: Books room, checks records, makes payment
- **Admin**: Searches records, verifies payments

Use Cases:

- Book Room
- View Booking Records
- Process Payment

Workflow Diagram

1. Launch App → Show Main Menu
2. Booking: Collect details → Validate → Assign room/customer ID → Save
3. Records: Search → Filter → Display data
4. Payment: Enter phone → Lookup → Pay → Update status

Sequence Diagram

Booking Flow:
User Inputs → App Validates → Auto-generate Room/ID → Save to Lists → Show Confirmation

Class/Component Diagram

- Main (Tkinter.Tk)
  - BookingScreen (Frame)
  - RecordScreen (Frame)
  - PaymentScreen (Frame)
  - Data (Parallel Python lists)

ER Diagram

No persistent storage used. If implemented: Customer, Booking, Payment tables related via foreign keys.

Design Decisions & Rationale

- **Tkinter** picked for widespread support, zero extra install for most users.
- Used **lists**, not databases: simplicity wins; easier setup, less that can break.
- Design: All-in-one window, not tabs. Fewer clicks—less confusion.
- Data validation strict—catches date/phone/incomplete fields early. Fewer surprises during use.
- Payments tracked by phone number: quick, easy. Not the safest, but practical for mini-hotels.

Implementation Details

The entire app coded in Python (about 400 lines). Used:

- `tkinter` for interface, buttons, and forms.
- `tkcalendar.DateEntry` for selecting dates.
- `random` module to create room numbers, customer IDs so users don't decide themselves.
- `datetime` used to subtract dates, get night counts.

Major Functions:

- `validate_date_obj`: Only allows year 2025-2027.
- `draw_home`: Main menu; lets users switch features.
- `draw_booking`: Collects, validates customer info—prevents duplicates/unpaid double bookings.
- `draw_record`: Text-based record viewer with instant search.
- `draw_payment`: Handles payment process.

Screenshots / Results

(Please insert actual screenshots in your Google Doc here. Show main menu, booking, records, payment.)

Testing Approach

Manual testing—no unit tests. Tried common scenarios:

- Valid bookings
- Duplicate phone
- Invalid dates
- Early check-out before check-in
- Payment re-entry
- Payment on completed bookings

Broke the app a few times (intentionally). Fixed bugs by adding checks.

Challenges Faced

- Tkinter's widget destruction sometimes glitches—needed lots of widget redrawing.
- Date validation tricky; tkcalendar gives string, had to ensure proper type with get_date().
- Tracking payment status. Hard to design payment system with no external backend.
- No persistence.

Learnings & Key Takeaways

- GUI with Tkinter is easier but limited.
- Validations are *everything* for real apps.
- Simpler code wins over over-complicated architectures in small projects.
- Polish matters—even tiny color tweaks made UI more fun to use.
- Sometimes the fastest backend is: no backend at all.

Future Enhancements

- Integrate SQLite for saving data.
- Add room categories (single, double, suite).
- Generate receipts as PDF after payment.
- Add basic authentication for staff/admin.
- Make responsive/adaptive UI for different screens.
- Enable emailing confirmation to customers.
- Introduce export: CSV, Excel of records.

- Add language options.

References

1. Python official documentation: https://docs.python.org/3/
2. Tkinter GUI Programming, TutorialsPoint: https://www.tutorialspoint.com/python/python_gui_programming.htm
3. tkcalendar widget PyPI: https://pypi.org/project/tkcalendar/
4. Custom project code and personal experimentation, 2025