Hotel Reservation System

Redesign and Rebuild System using Microservices and Multi-Modal Clients

1. Requirements:

Description with UI sketch of all use cases. Use cases should be described with the standard format including a short description and a sequence of user actions and system reactions. Each user action or system reaction might include a UI sketch. You should include all use cases, both from Iterations 1 and 2 and the new ones for Iteration 3.

Use Case 1 : Login into the Hotel Reservation System

Actor: Receptionist/User, Admin

Goal: Actor wants to login into the Hotel Reservation System

Preconditions: Not applicable

Summary: The login page serves as the entry point for the user into the system.
The homepage serves as the main interface for users to navigate through the hotel reservation system. It provides easy access to crucial functions such as booking, managing reservations, accessing room details, and handling guest information. The presence of user-specific information helps improve the experience, while the room availability and reservation status gives users a quick overview of the hotel's room reservation availability.

Functionalities available for Admin:

- Room Booking - To book a stay
- Cancel Booking - To cancel a reservation
- Rooms - Check availability of rooms
- Check Out - Check out a reservation
- Guests - List of guests staying at the hotel
- Add, Delete and View Users - Add new users, delete existing users and view the list of all available users

Functionalities available for Receptionist/User: (Newly added use cases for receptionist in this iteration)

- Room Booking - To book a stay
- Cancel Booking - To cancel a reservation
- Check Out - Check out a reservation
- Guests - List of guests staying at the hotel.

Steps:

User Actions	System Responses
1. User enters username	3. The system takes user input. The login username is varchar and password is set to integer.
2. User enters password	
4. User clicks “Login”	5. The system checks the data input by the user and verifies it with the redis. If the user is present, it logs the user into the system and displays the homepage, else it displays an error dialog showing incorrect user credentials.

UI Sketch for Desktop Based Client:

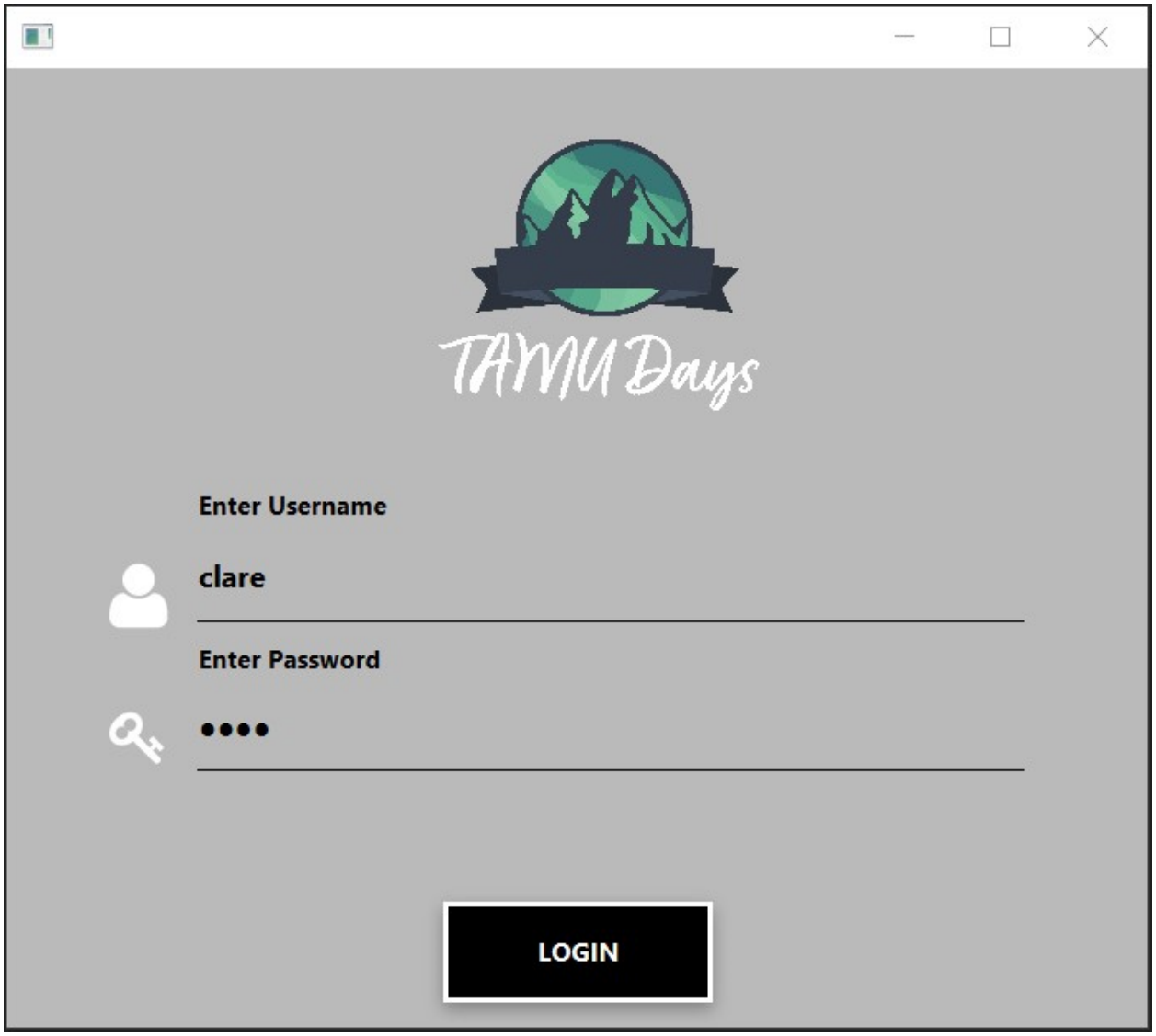


Figure 1. Login Screen.

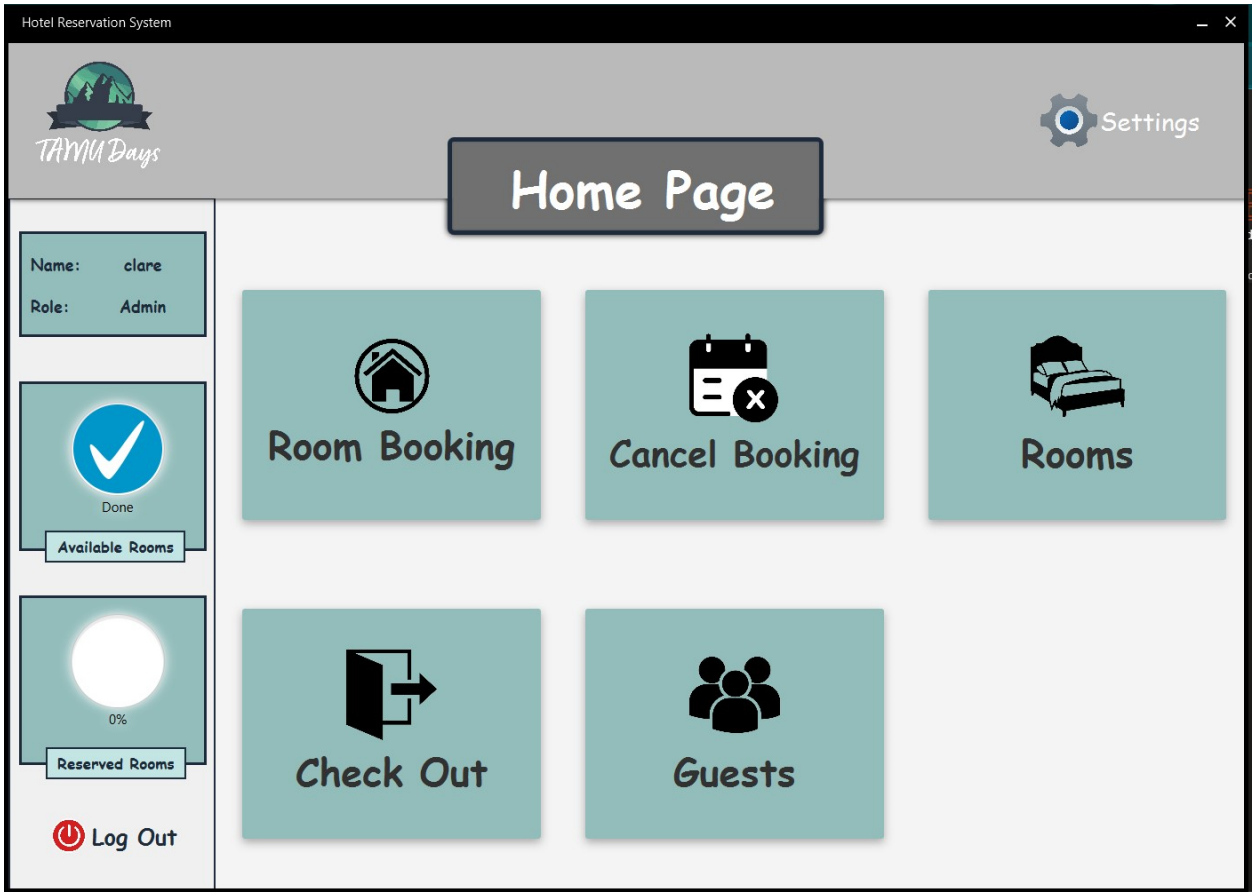


Figure 2. Home Page for Admin

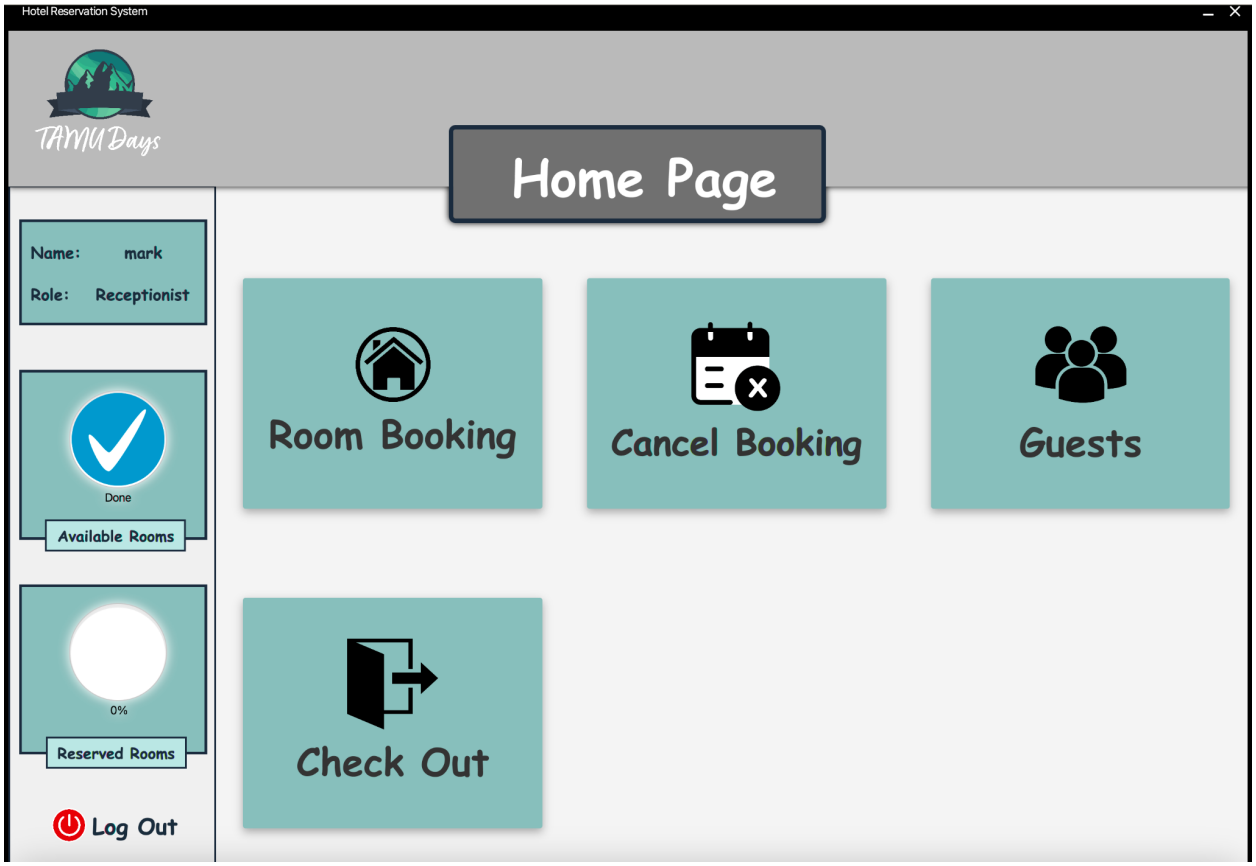


Figure 3. Home Page for Receptionist/User

UI Sketch for Web Based Client:

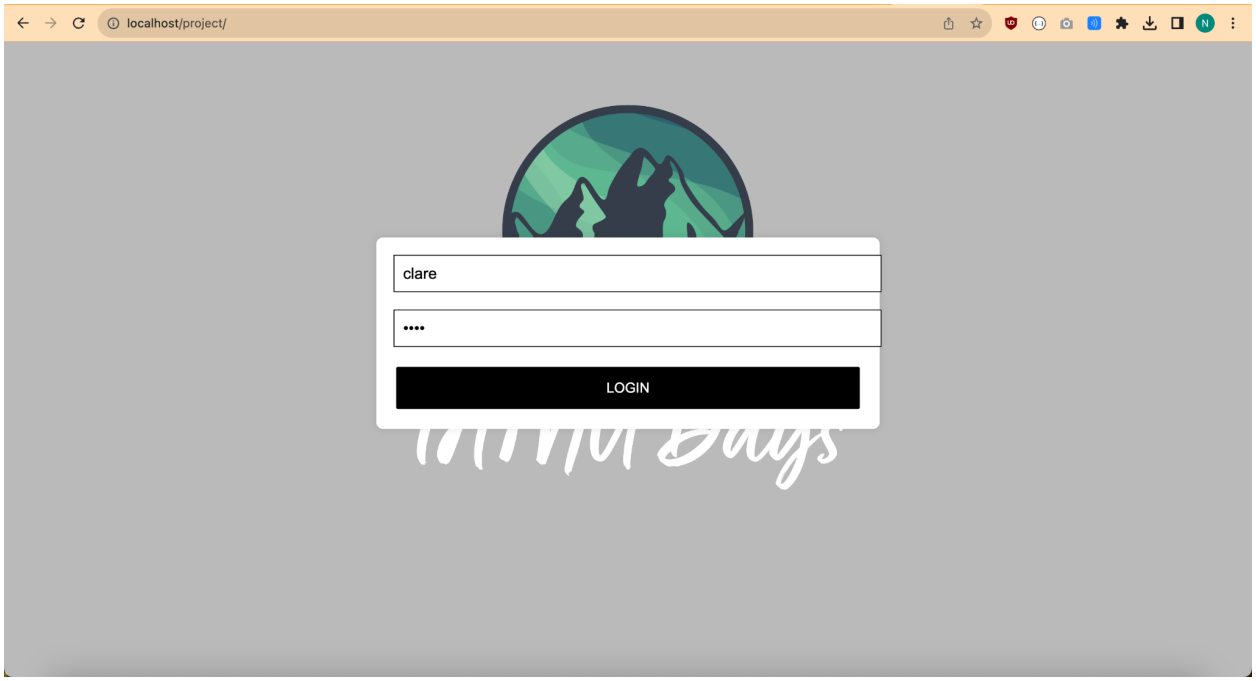


Fig 4. Login Screen

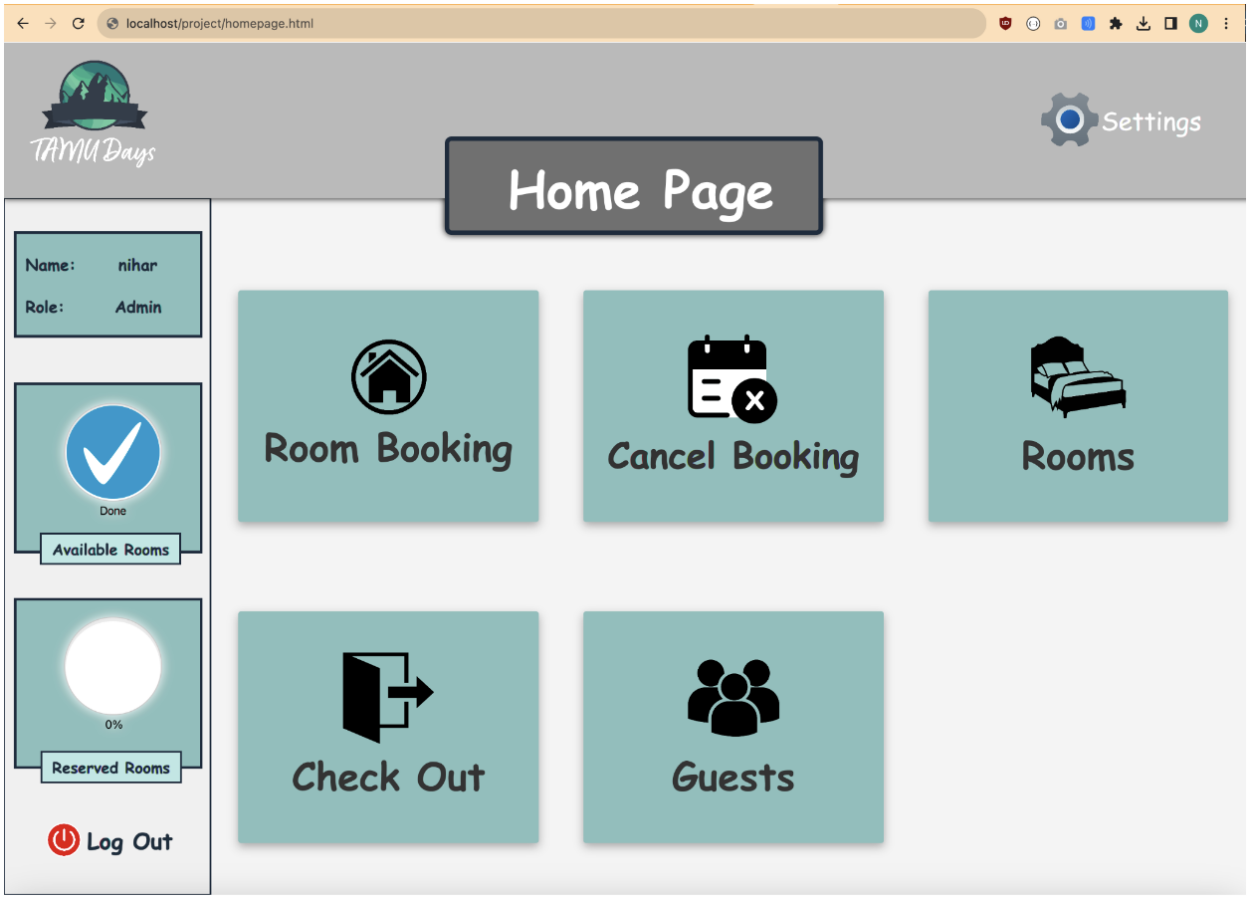


Fig 5. Home Page

Use Case 2 : Booking and room reservation

Actor: Receptionist/User

Goal: Book and reserve room for the duration of stay

Preconditions: Receptionist/User must be logged in the system

Summary: The room booking page is designed to collect and store guest information like name, phone, email, address, city, nationality, passport number, credit card details, room type, room capacity, check in and checkout date necessary for reserving a room and processing payment. The hotel system has three different types of rooms with three types of capacity. Guests can choose their room preference and their desired date for booking. The navigation sections on the left provide easy access to other areas of the system, allowing guests to manage their reservations and explore available rooms and other features. It is designed to help users make their reservations after filling all the necessary information. On clicking the clear button, the data input fields are cleared for fresh input. After clicking the search button, the user is assigned a room number for their stay. The submit button generates a room booking confirmation prompt with all booking information of the user. The admin can verify the guest information. The user can then press Book and confirm their booking.

Steps:

User Actions	System Responses
1. User enters all input fields for room reservation	2. The system has input types set for each type of input in the user information for room reservation.
3. User clicks “Search”	4. The system assigns a room ID to the user reservation
5. User clicks “Clear”	6. The system clears all the filled input fields and makes way for a fresh input for the reservation page.
7. User click “Submit”	8. The system displays a booking confirmation page showing the user all the filled input information.
9. User clicks “Back”	10. The system takes the user to the

	homepage.
11. User clicks “Book” on confirmation page	12. The system adds the user data into the mongo db collection and confirms the room booking. The room number is then marked as occupied and the guest list is updated with the user data.
13. User clicks “Cancel” on confirmation page	14. The system reverts back to the room reservation page.

UI Sketch for Desktop Based Client:

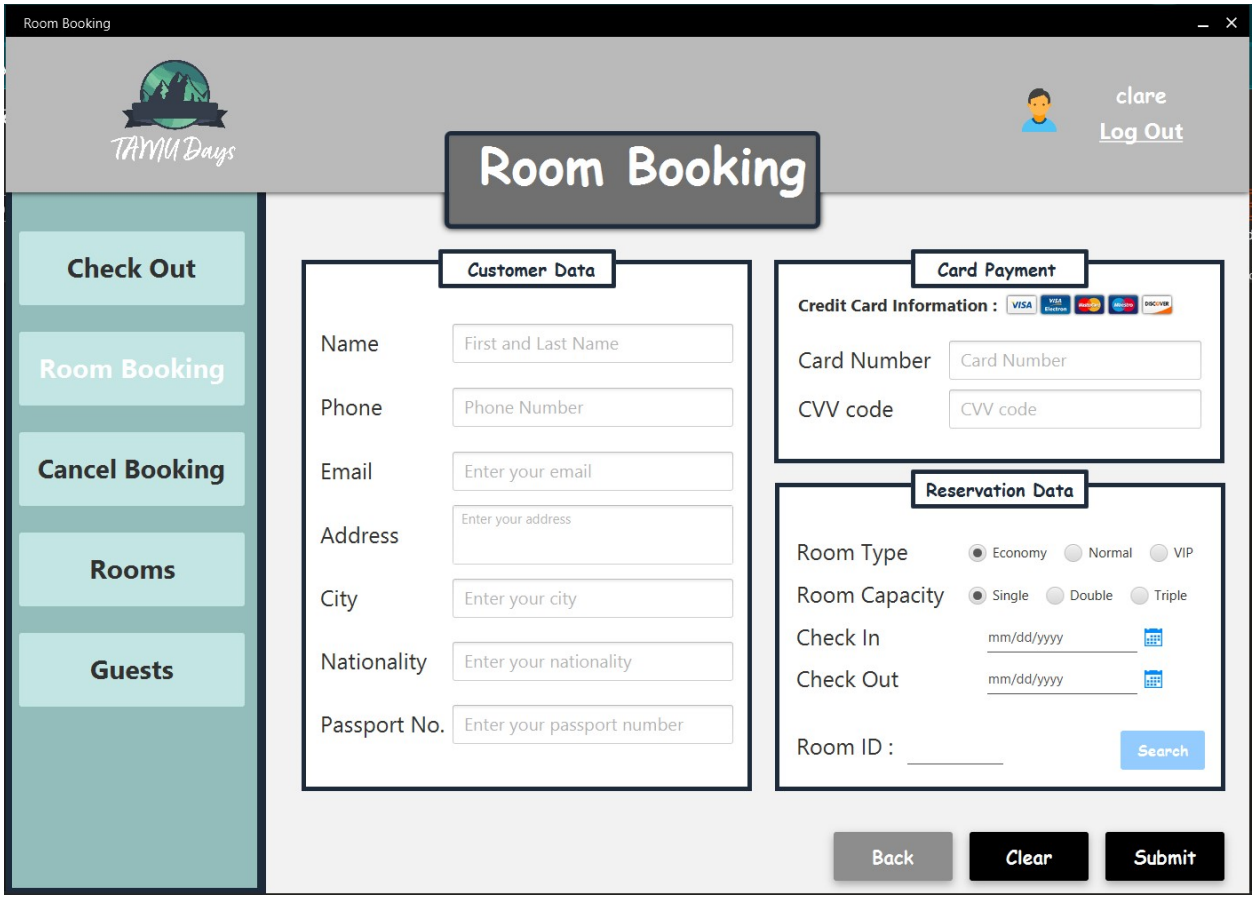


Figure 6. Room Booking Screen

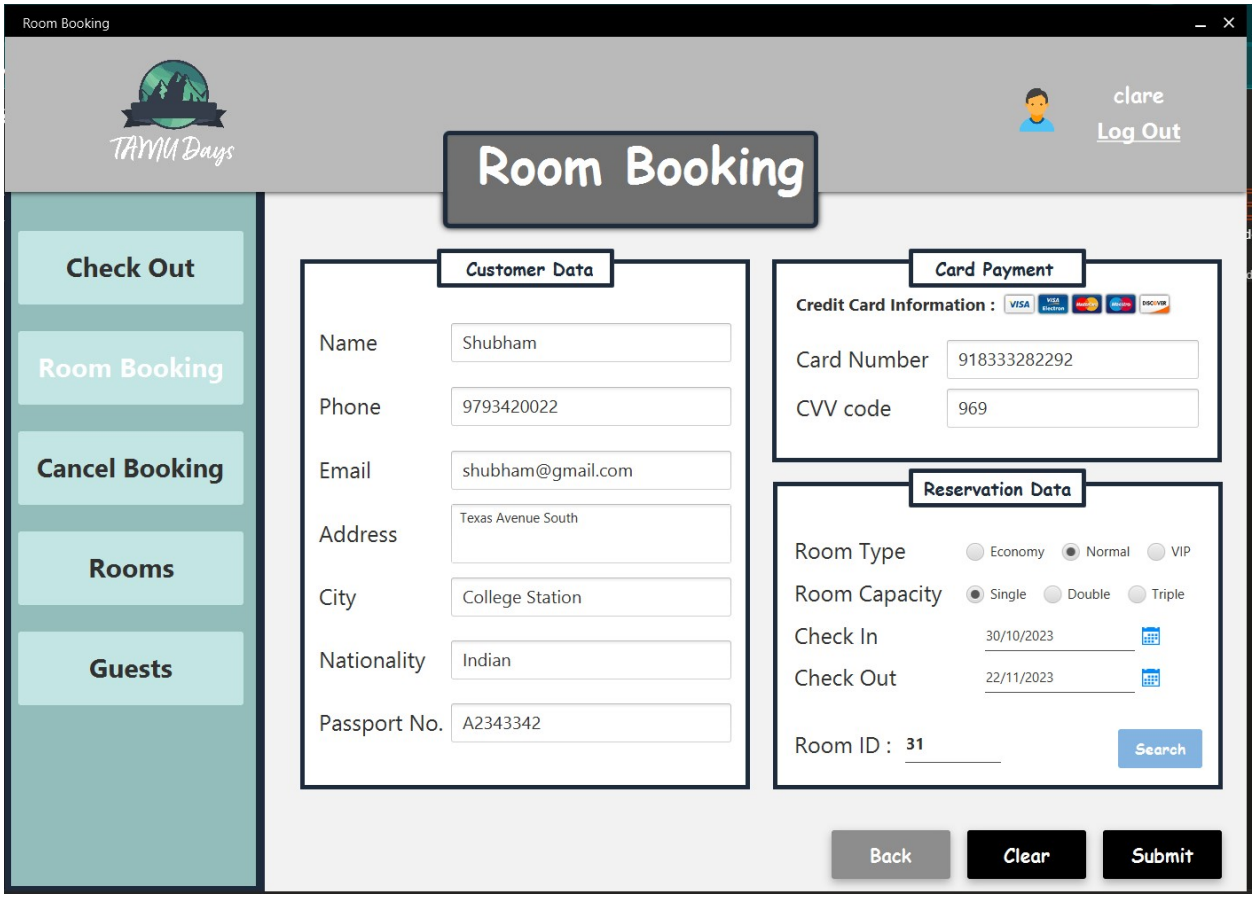


Figure 7. Room Booking Input Filled

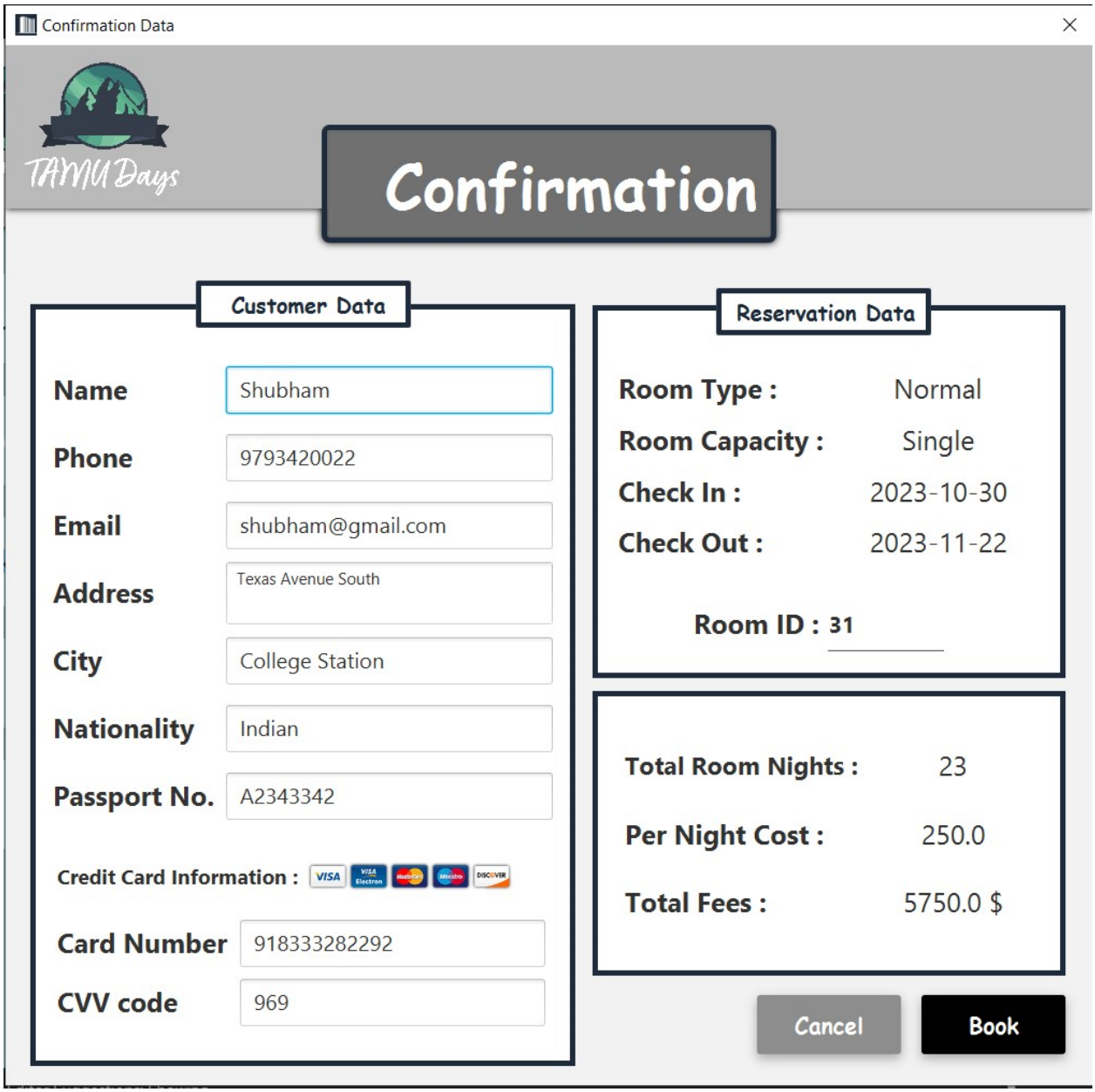


Figure 8. Room Booking Confirmation

UI Sketch for Web Based Client:

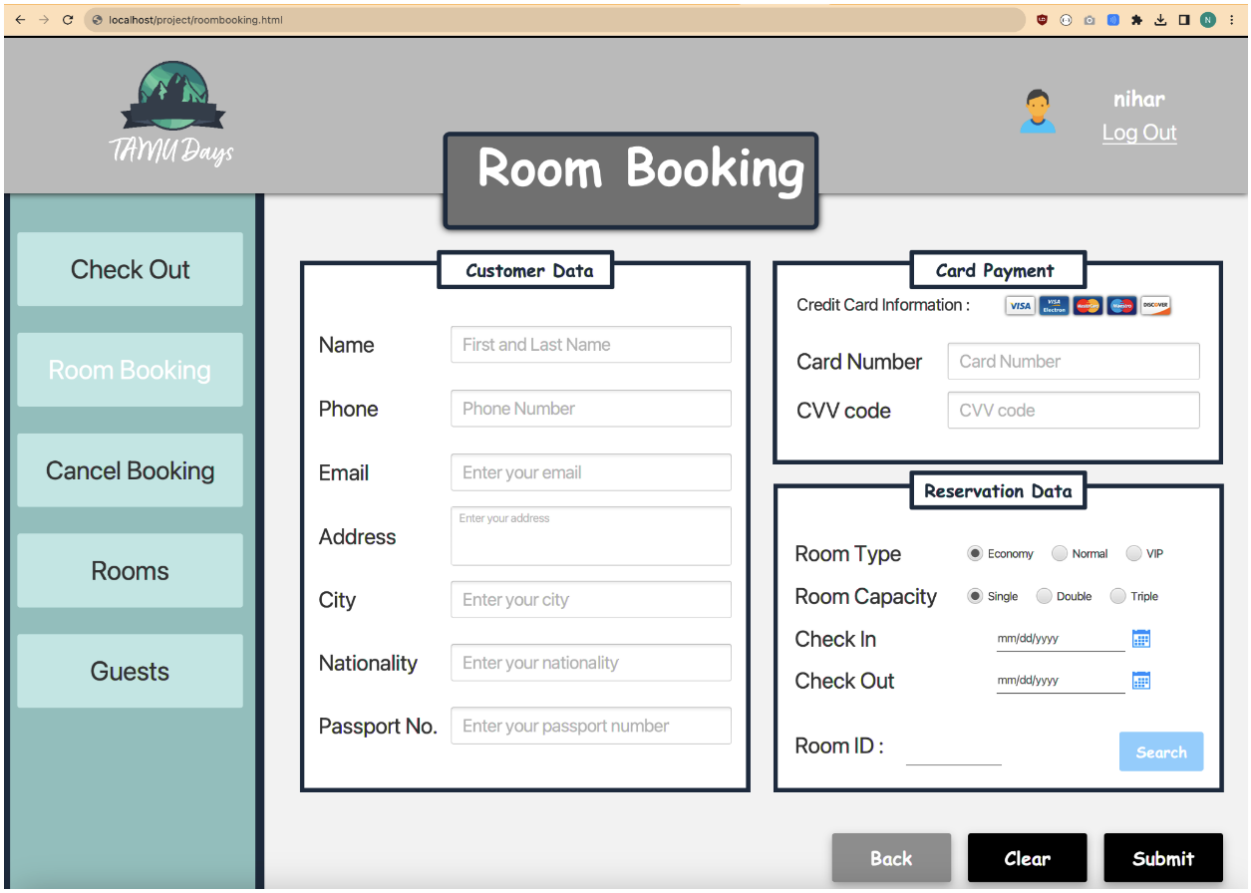


Fig 9. Room Booking Input Filled

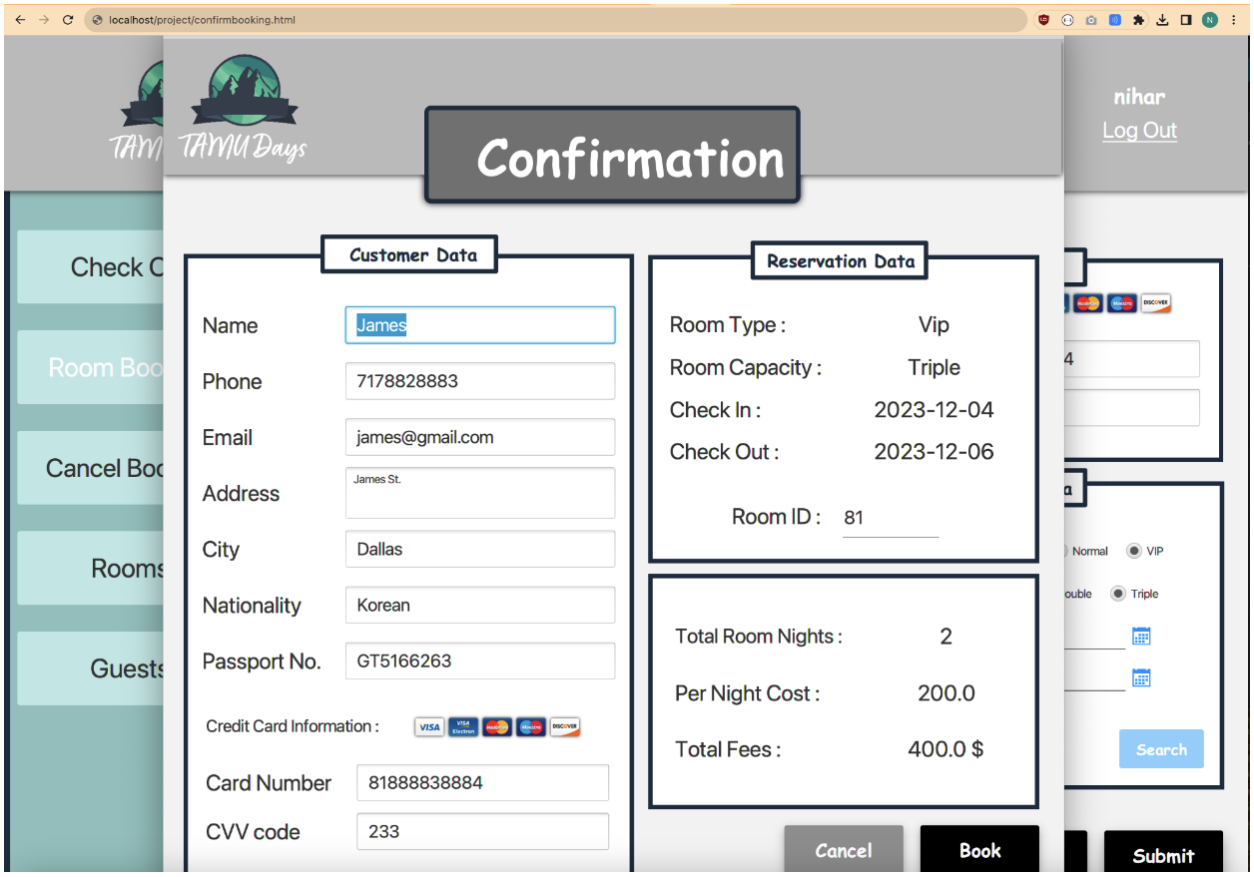


Fig 10. Room Booking Confirmation

Use Case 3 : Cancellation of room booking

Actor: Receptionist/User, Admin

Goal: Cancel the booked room

Preconditions: The user must be logged into the system and room should be booked in order to not receive an error message

Summary: The cancel booking page is designed to allow users to cancel their reservations conveniently. The navigation sections on the left provide easy access to other areas of the system, allowing users to manage their reservations, explore available rooms, and access guest-related features. Administrators can also use this page to assist guests with cancellations if needed. When the current user of the system clicks on the “Cancel Booking” button, the room reserved becomes available and the guest entry is removed from the mongo db collection. This helps keep a track of the current and future active reservations.
After the user details have been removed from the collection, a message prompt displays successful cancellation.

Steps:

User Actions	System Responses
1. User enters the room number to cancel the reservation	2. The system is designed to check for the validation that if the user has already checked in to the hotel, the cancellation cannot be done. If there is still time for the check in date to arrive, the user can successfully cancel his reservation.
3. User clicks “Clear”	4. The system clears the room number input field.
5. User clicks “Cancel Booking”	6. The system verifies the room number with the room ID in the database, then deletes the reservation from the mongo db collection and assigns the room as free for booking in the rooms list. The guest list is updated after successful cancellation. A prompt displays successful cancellation to the user.

7. User clicks “Back”	8. The system directs to the homepage.

UI Sketch for Desktop Based Client:

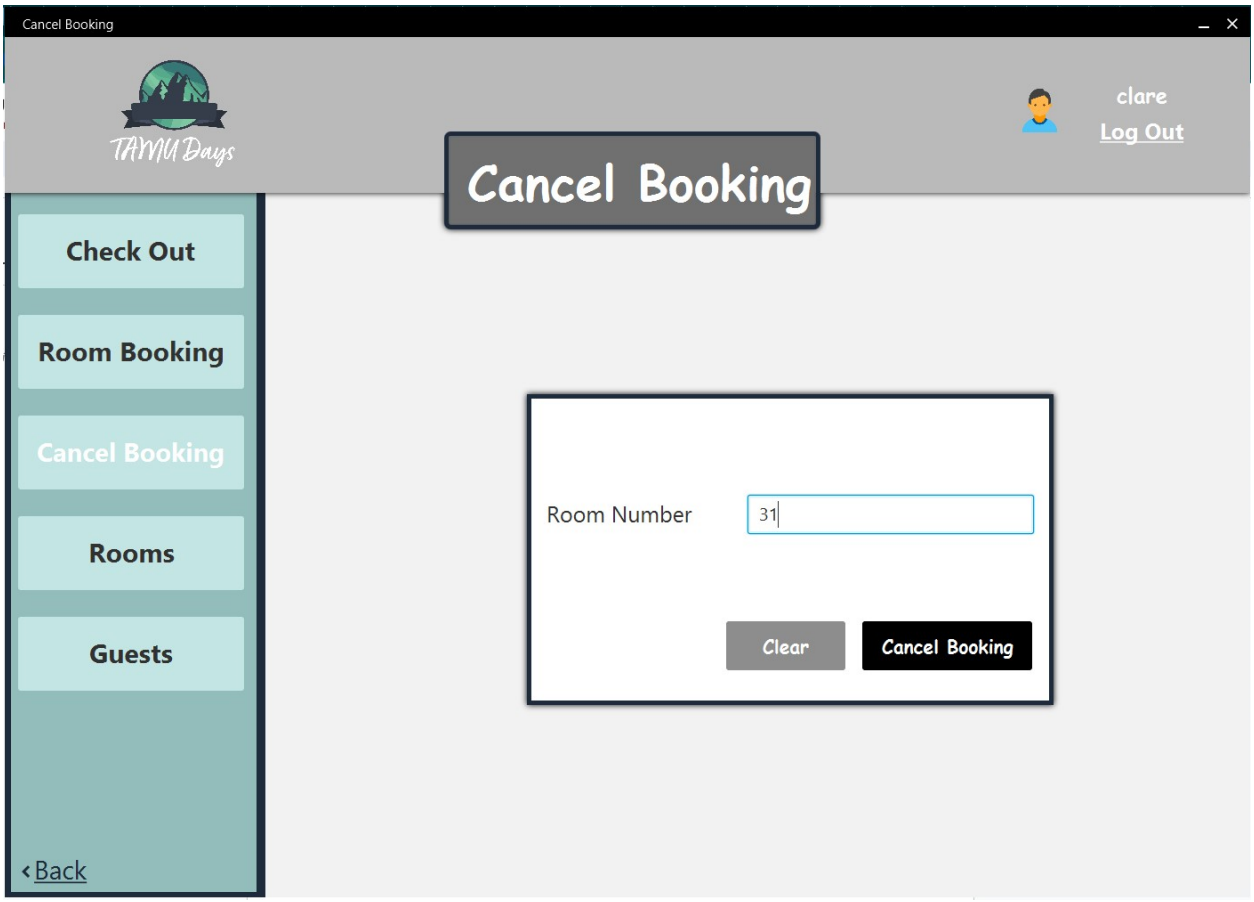


Figure 11. Cancel Booking Screen

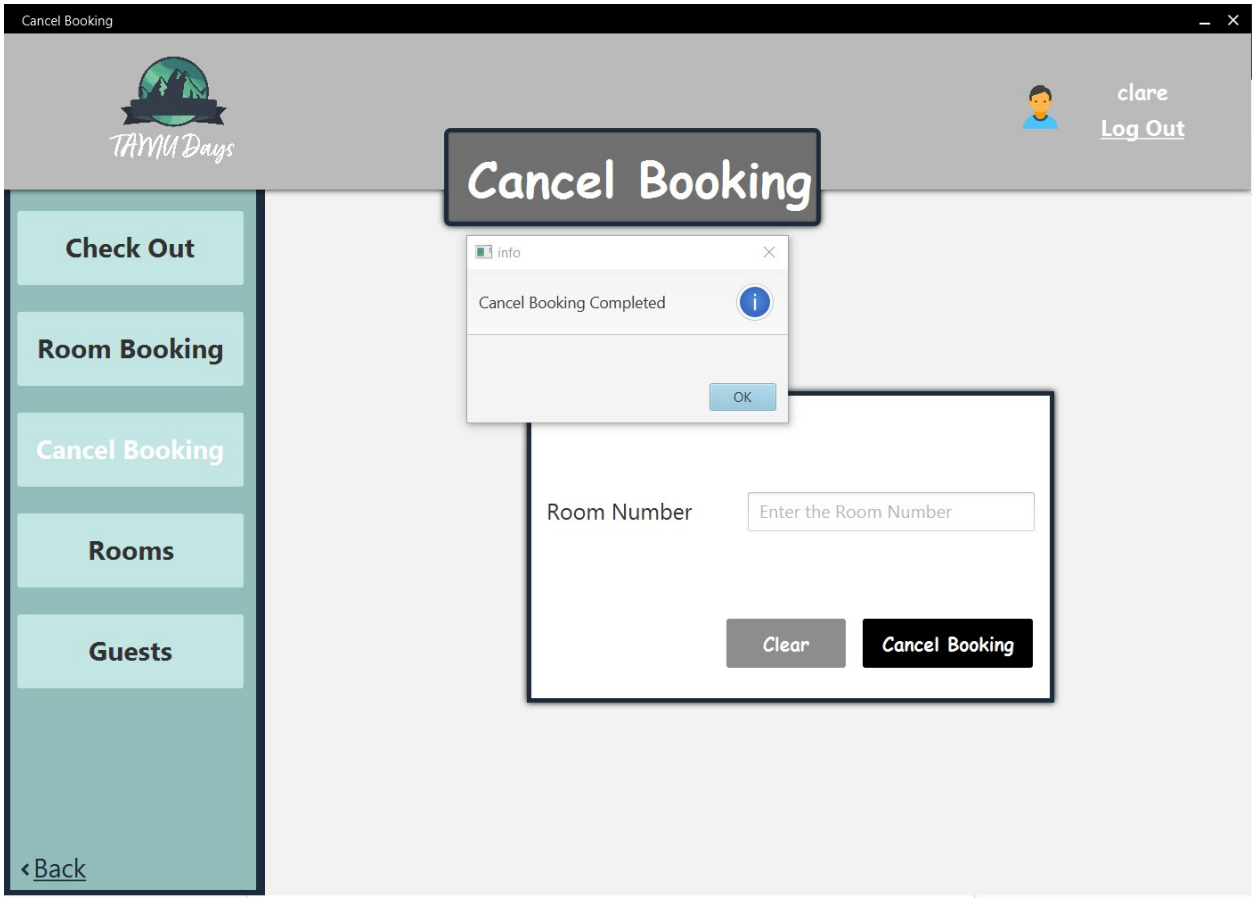


Figure 12. Booking Cancellation Confirmation Prompt

UI Sketch for Web Based Client:

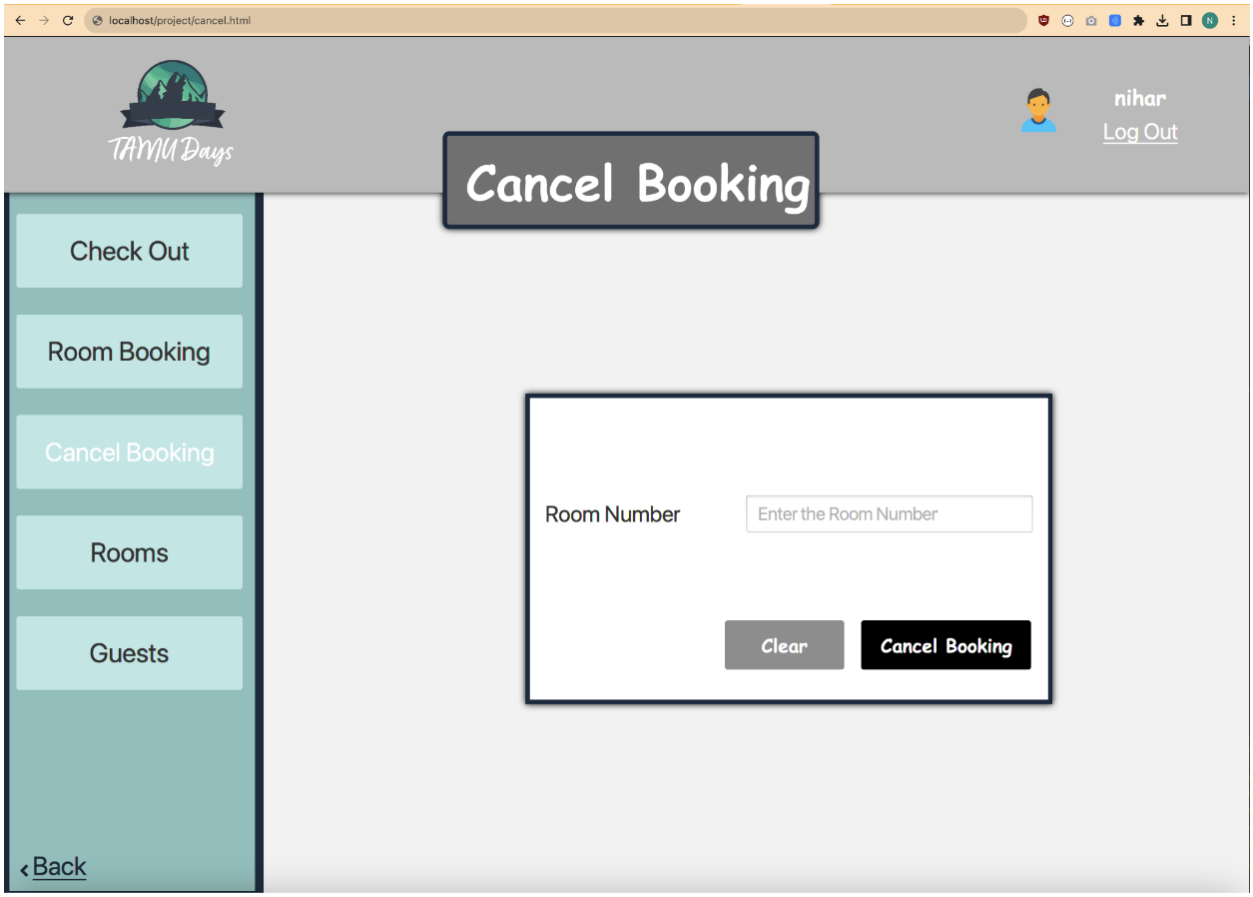


Fig 13. Cancel Booking Screen

Use Case 4 : Room occupancy monitor for the administrator

Actor: Admin

Goal: Monitor the availability of rooms and check the booked rooms

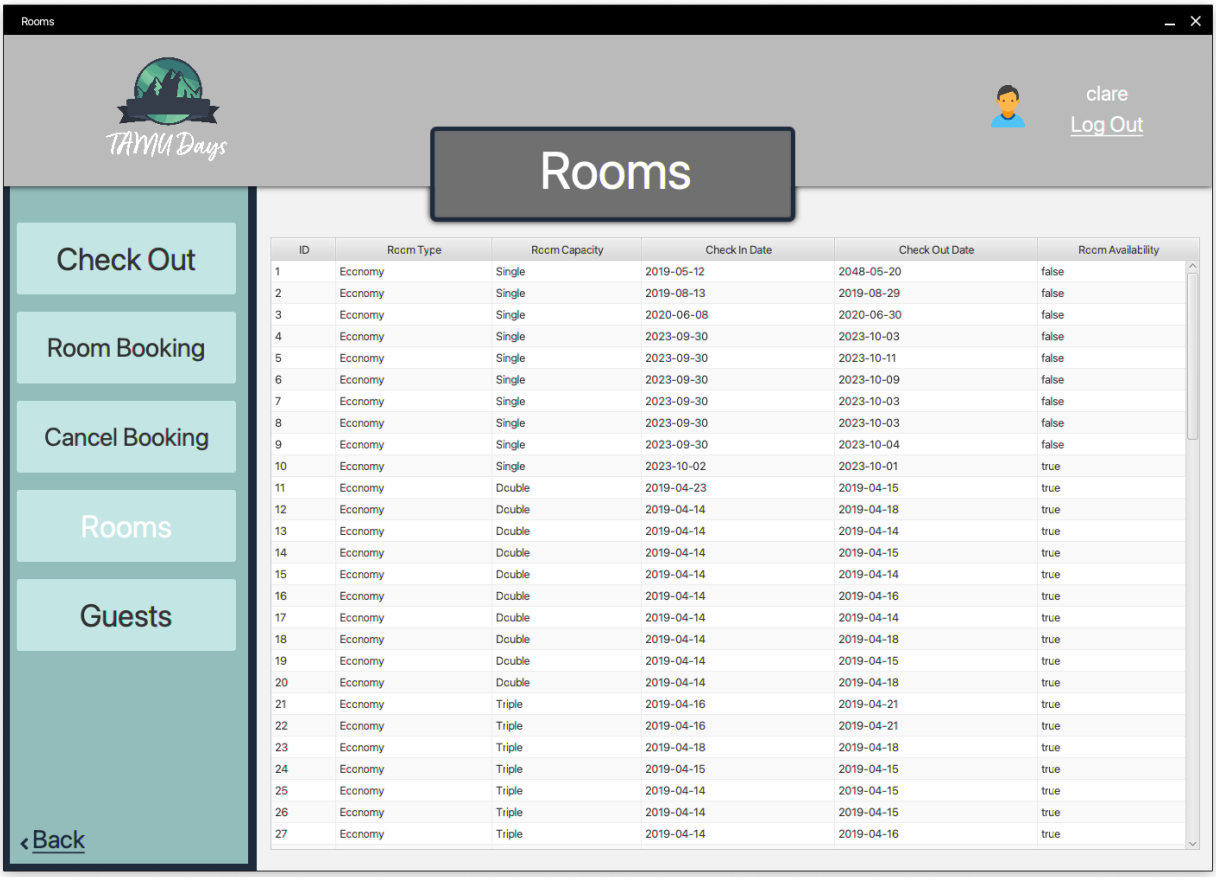
Preconditions: Admin must be logged into the system

Summary: The "Rooms" page serves as a valuable resource for users to explore the available rooms, their types, and occupancy status. The admin can check the room availability if customers use the walkin booking option. The admin can also check and confirm a room's availability to book for customers calling on the hotel service contact number. This use case is convenient for getting information about available rooms and then booking them for guests in real time without using a guest login.
The navigation sections on the left provide easy access to other areas of the system, enabling users to manage bookings, explore room options, and access guest-related features.

Steps:

User Actions	System Responses
1. User clicks "Rooms" on the homepage	2. The system fetches the room details like type, capacity, checkin, checkout and occupancy status of all the available rooms in the hotel from the database.

UI Sketch for Desktop Based Client:



ID	Room Type	Room Capacity	Check In Date	Check Out Date	Room Availability
1	Economy	Single	2019-05-12	2048-05-20	false
2	Economy	Single	2019-08-13	2019-08-29	false
3	Economy	Single	2020-06-08	2020-06-30	false
4	Economy	Single	2023-09-30	2023-10-03	false
5	Economy	Single	2023-09-30	2023-10-11	false
6	Economy	Single	2023-09-30	2023-10-09	false
7	Economy	Single	2023-09-30	2023-10-03	false
8	Economy	Single	2023-09-30	2023-10-03	false
9	Economy	Single	2023-09-30	2023-10-04	false
10	Economy	Single	2023-10-02	2023-10-01	true
11	Economy	Double	2019-04-23	2019-04-15	true
12	Economy	Double	2019-04-14	2019-04-18	true
13	Economy	Double	2019-04-14	2019-04-14	true
14	Economy	Double	2019-04-14	2019-04-15	true
15	Economy	Double	2019-04-14	2019-04-14	true
16	Economy	Double	2019-04-14	2019-04-16	true
17	Economy	Double	2019-04-14	2019-04-14	true
18	Economy	Double	2019-04-14	2019-04-18	true
19	Economy	Double	2019-04-14	2019-04-15	true
20	Economy	Double	2019-04-14	2019-04-18	true
21	Economy	Triple	2019-04-16	2019-04-21	true
22	Economy	Triple	2019-04-16	2019-04-21	true
23	Economy	Triple	2019-04-18	2019-04-18	true
24	Economy	Triple	2019-04-15	2019-04-15	true
25	Economy	Triple	2019-04-14	2019-04-15	true
26	Economy	Triple	2019-04-14	2019-04-15	true
27	Economy	Triple	2019-04-14	2019-04-16	true

Figure 14. Room Occupancy Info Screen

UI Sketch for Web Based Client:

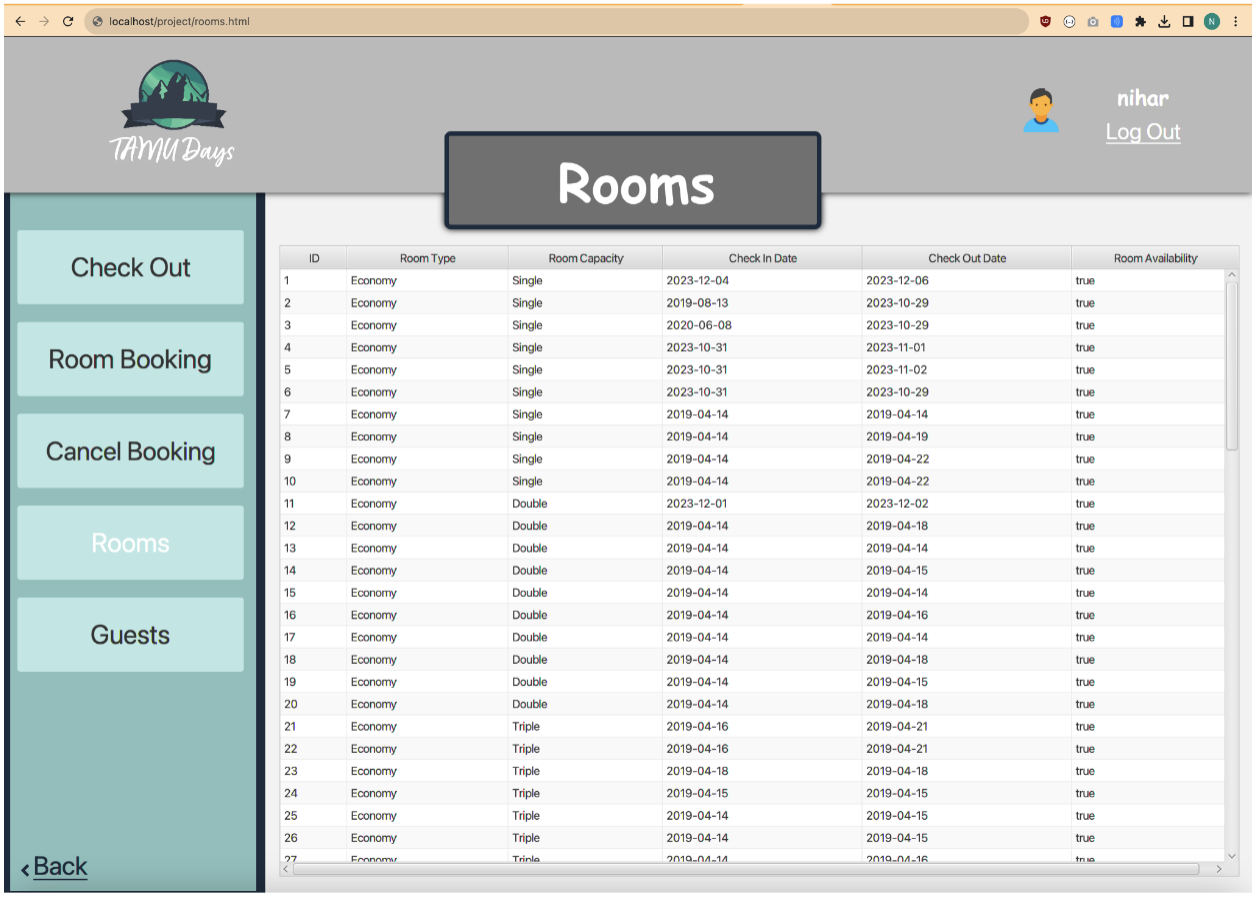


Fig 15. Room Occupancy Info Screen

Use Case 5 : Guest list

Actor: Admin

Goal: The admin can find the reservation details of the guests currently booked with the hotel.

Preconditions: Admin must be logged into the system. The guests should have booked the rooms in the hotel.

Summary: The "Guests" page serves as a valuable resource for users to confirm their reservation details. The admin can check the room numbers of the checked in guests and use it to check them out. This use case is useful for getting information about users not checked in but wanting to cancel their booking.
The navigation sections on the left provide easy access to other areas of the system, enabling users to manage bookings, explore room options, and access guest-related features.

Steps:

User Actions	System Responses
1. User clicks “Guests” on the homepage	2. The system fetches the guest details like room ID, Name, Email, Address,.. and total bill for all the guests booked in the hotel from the mongo db collection.
3. User clicks “Back”	4. The system directs the user to the homepage.

UI Sketch for Desktop Based Client:

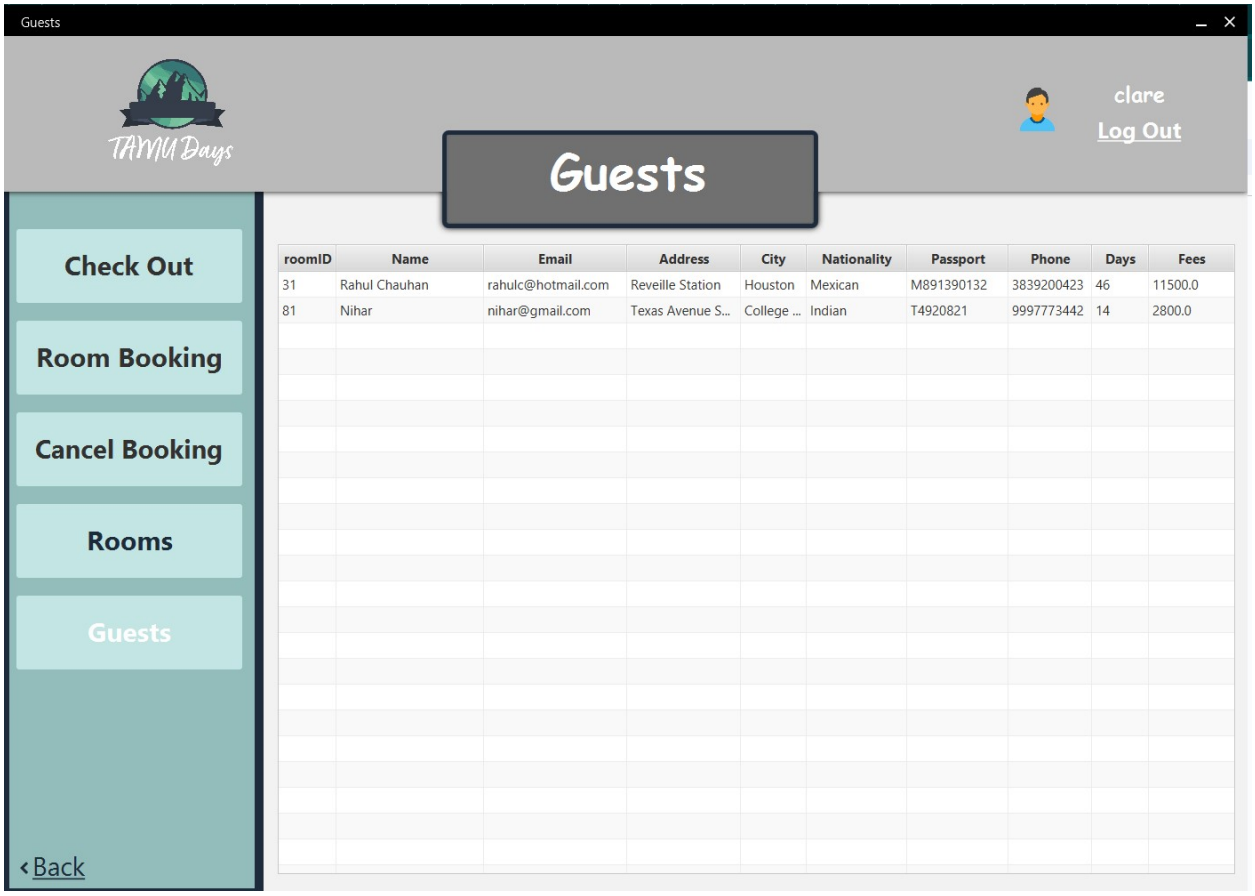


Figure 16. Guest List

UI Sketch for Web Based Client:

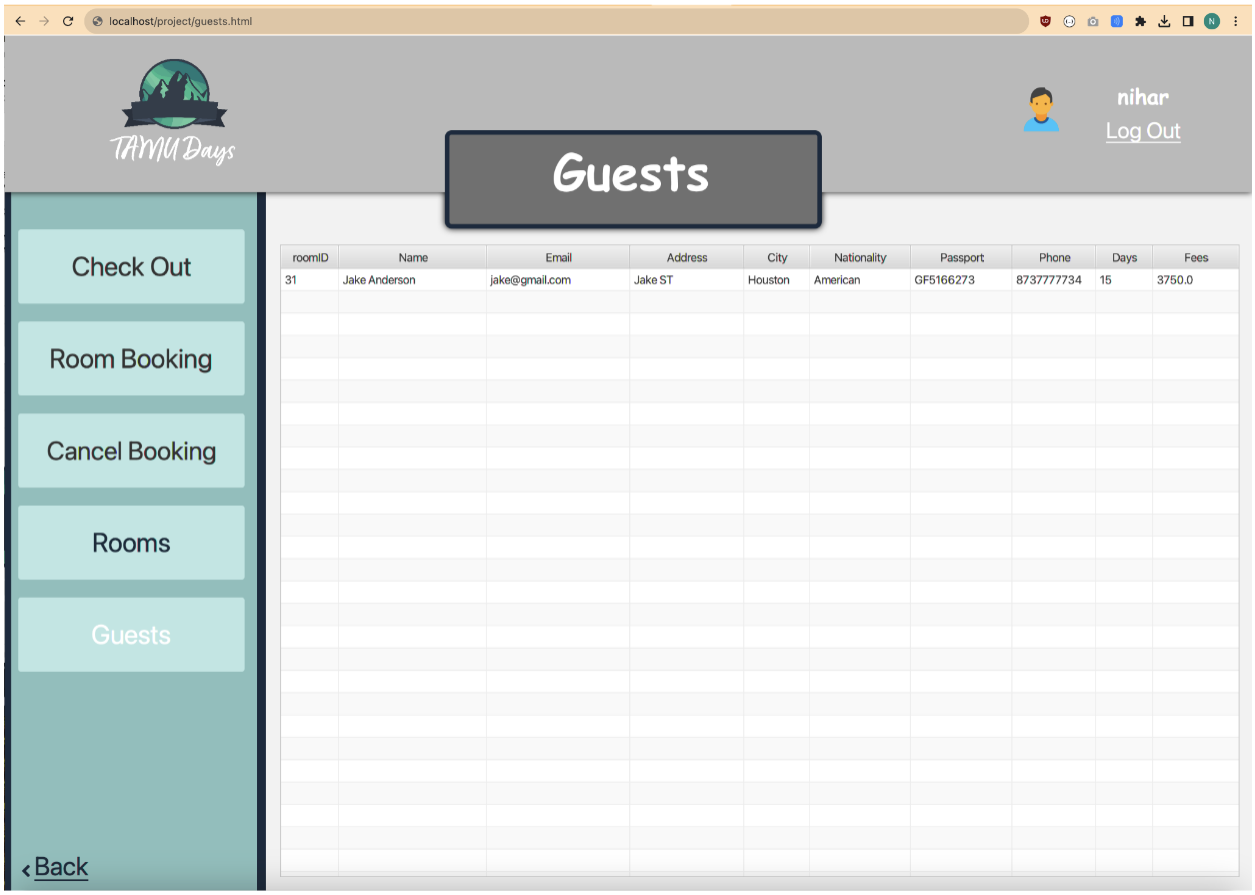


Fig 17. Guest List

Use Case 6 : Checkout Booking

Actor: Receptionist/User, Admin

Goal: To checkout from the hotel room

Preconditions: The user must be checked in the hotel room.

Summary: The Check out booking page is designed to allow users to checkout their reservations conveniently. The navigation sections on the left provide easy access to other areas of the system, allowing users to manage their reservations, explore available rooms, and access guest-related features. When the current user of the system clicks on the “Check out” button, the room reserved becomes available and the guest entry is removed from the mongo db collection. This helps keep a track of the current and future active reservations.

After the user details have been removed from the collection, a message prompt displays successful checkout confirmation.

Steps:

User Actions	System Responses
1. User inputs room number to checkout a booking	2. The system is designed to check for the validation that if the user has already checked in to the hotel, only then a booking would be marked as checked out. If any guest has a future booking in the hotel, that booking cannot be marked as checked out rather the user could cancel that booking.
3. User clicks “Clear”	4. The system clears the room number input field.
5. User clicks “Check Out”	6. The system verifies the room number with the room ID in the database, then deletes the reservation from the mongo db collection and assigns the room as free for booking in the rooms list. The guest list is updated after successful

	cancellation. A prompt displays successfully checked out booking to the user.
7. User clicks “Back”	8. The system directs to the homepage.

UI Sketch for Desktop Based Client:

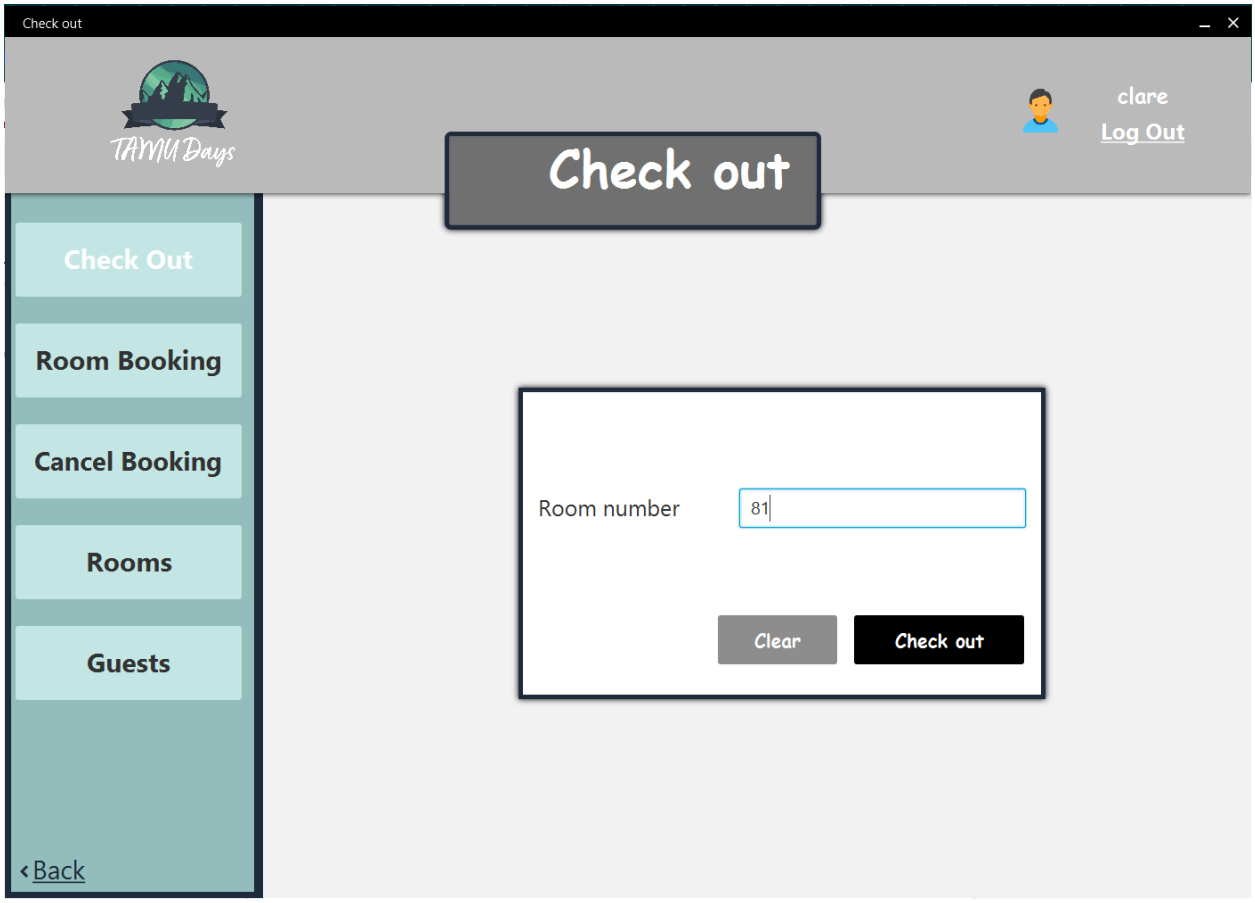


Figure 18. Checkout booking screen

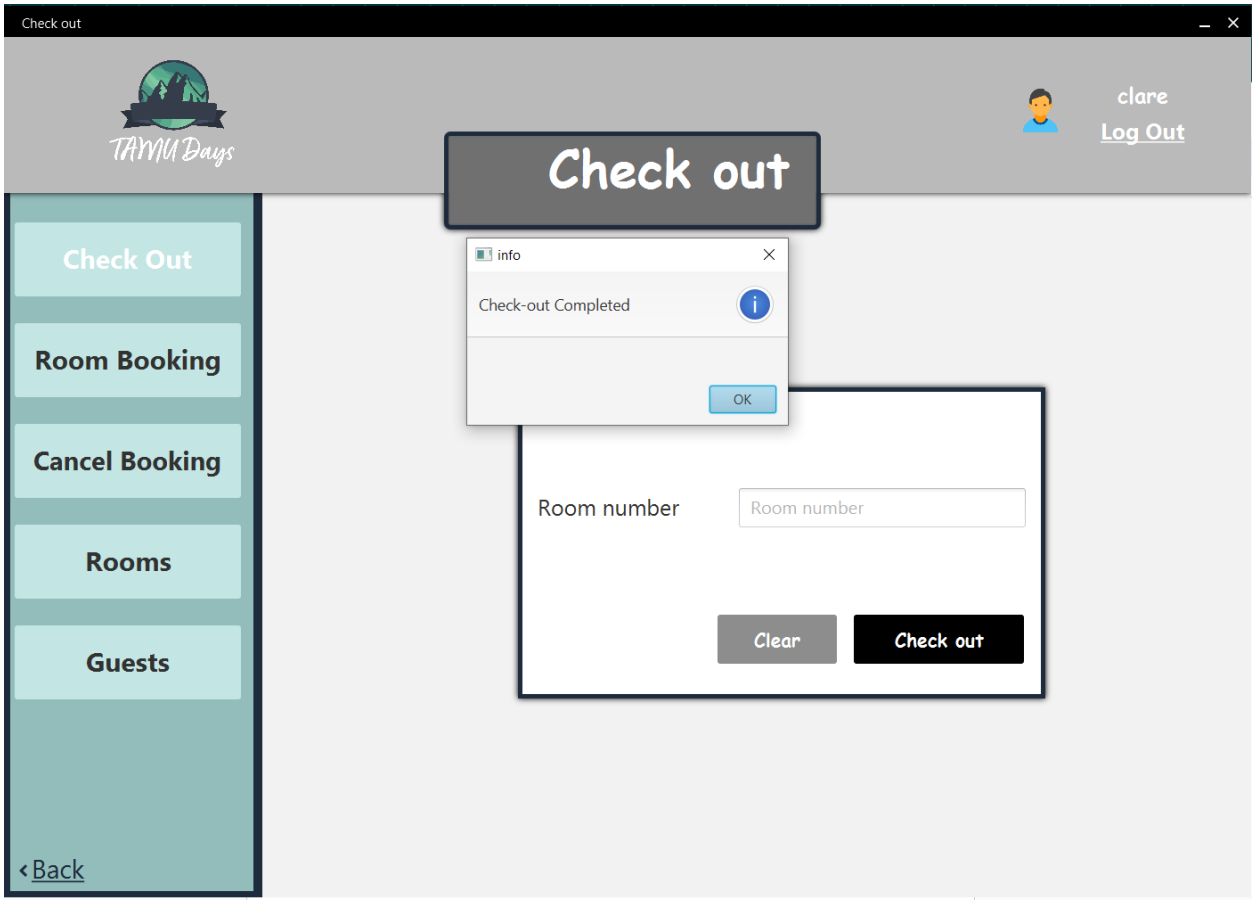


Figure 19. Checkout booking successful prompt

UI Sketch for Web-Based Client

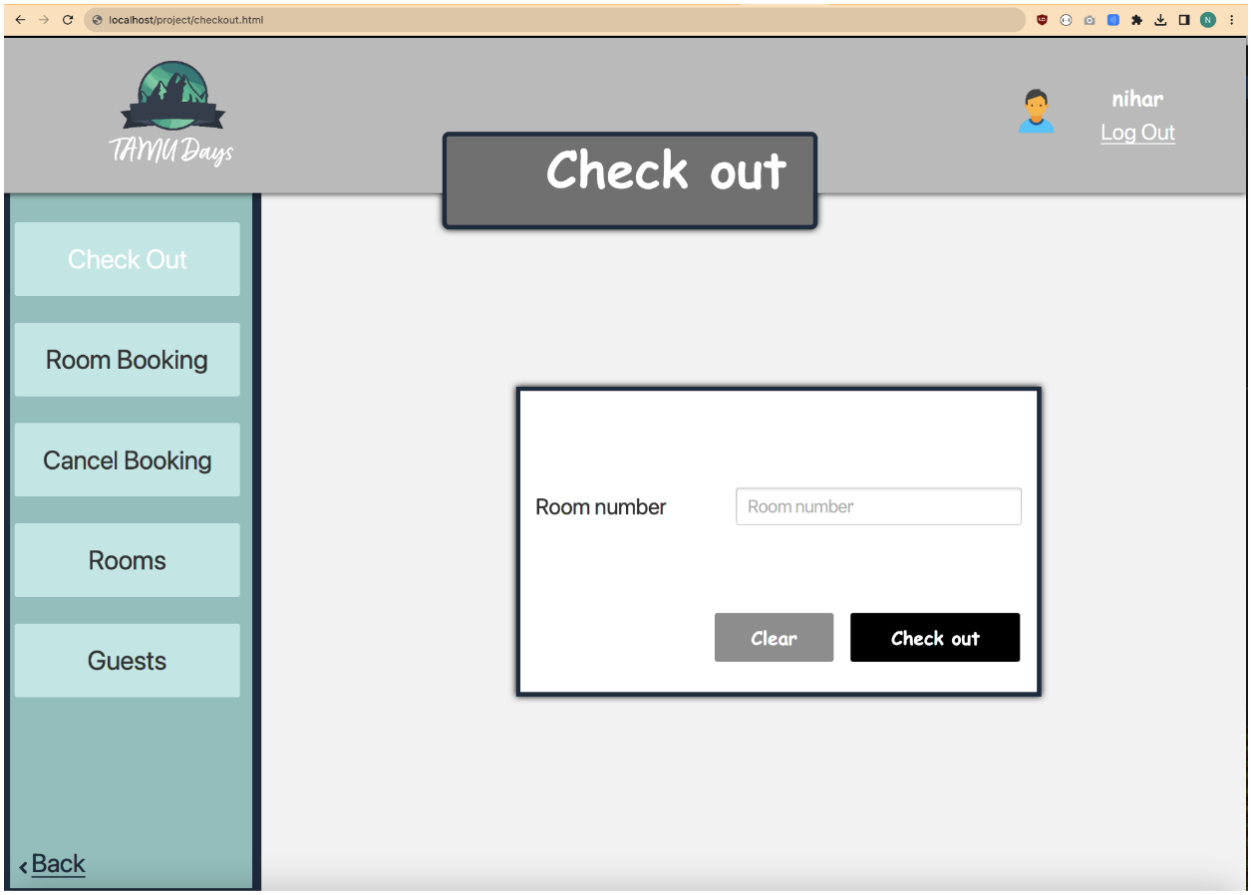


Fig 20. Checkout booking screen

Use Case 7 : Administrator settings - View

Actor: Admin

Goal: To view the current users of the hotel system

Preconditions: The admin must be logged into the system

Summary: This functionality is used to view all registered users by the logged in user. Logged in users can view details of the registered user such as their username, password and their admin status. Admin status column states whether the user is admin or not.

Steps:

User Actions	System Responses
1. User clicks “View All Users”	2. System calls “View All Users Page” showing all the registered users.

UI Sketch for Desktop Based Client:

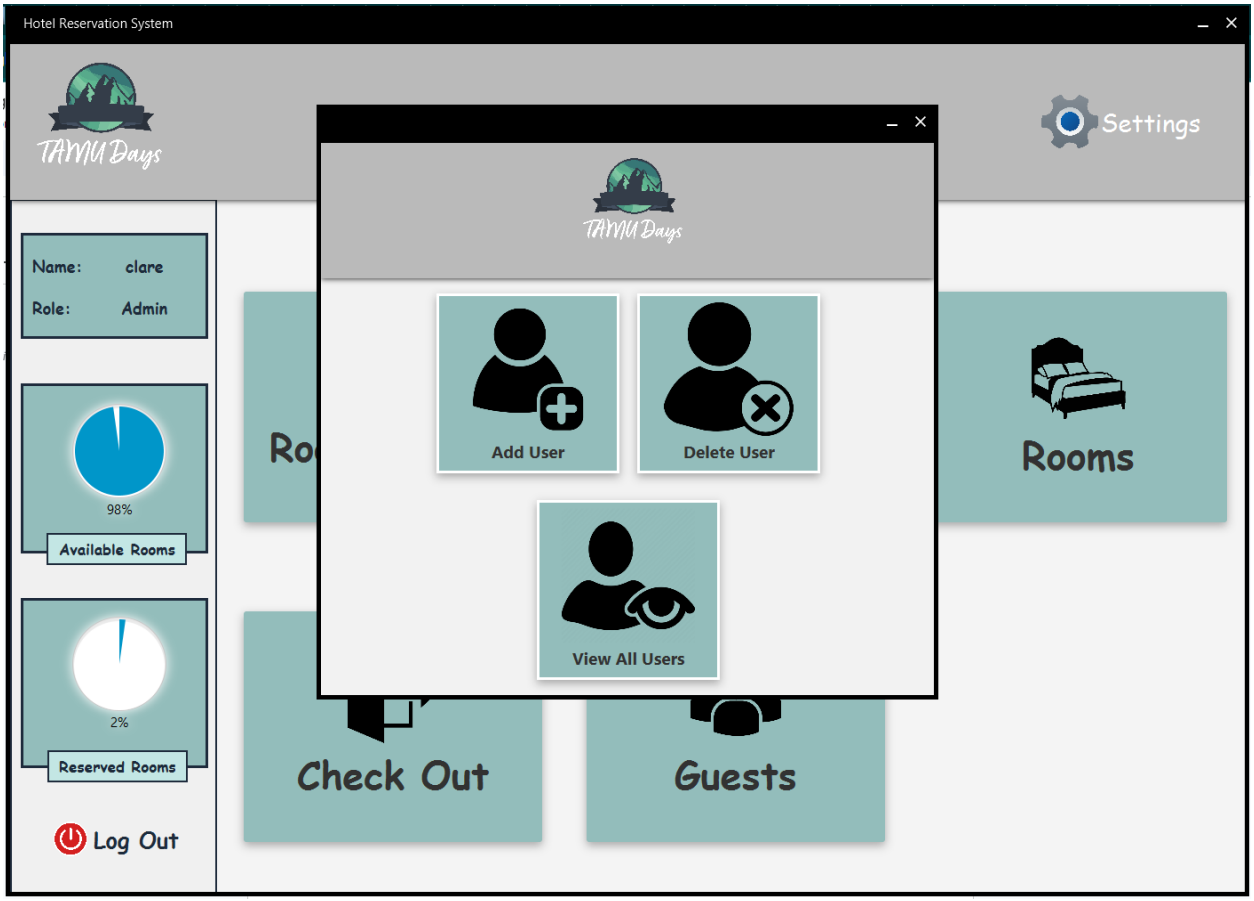


Figure 21. Administrator settings

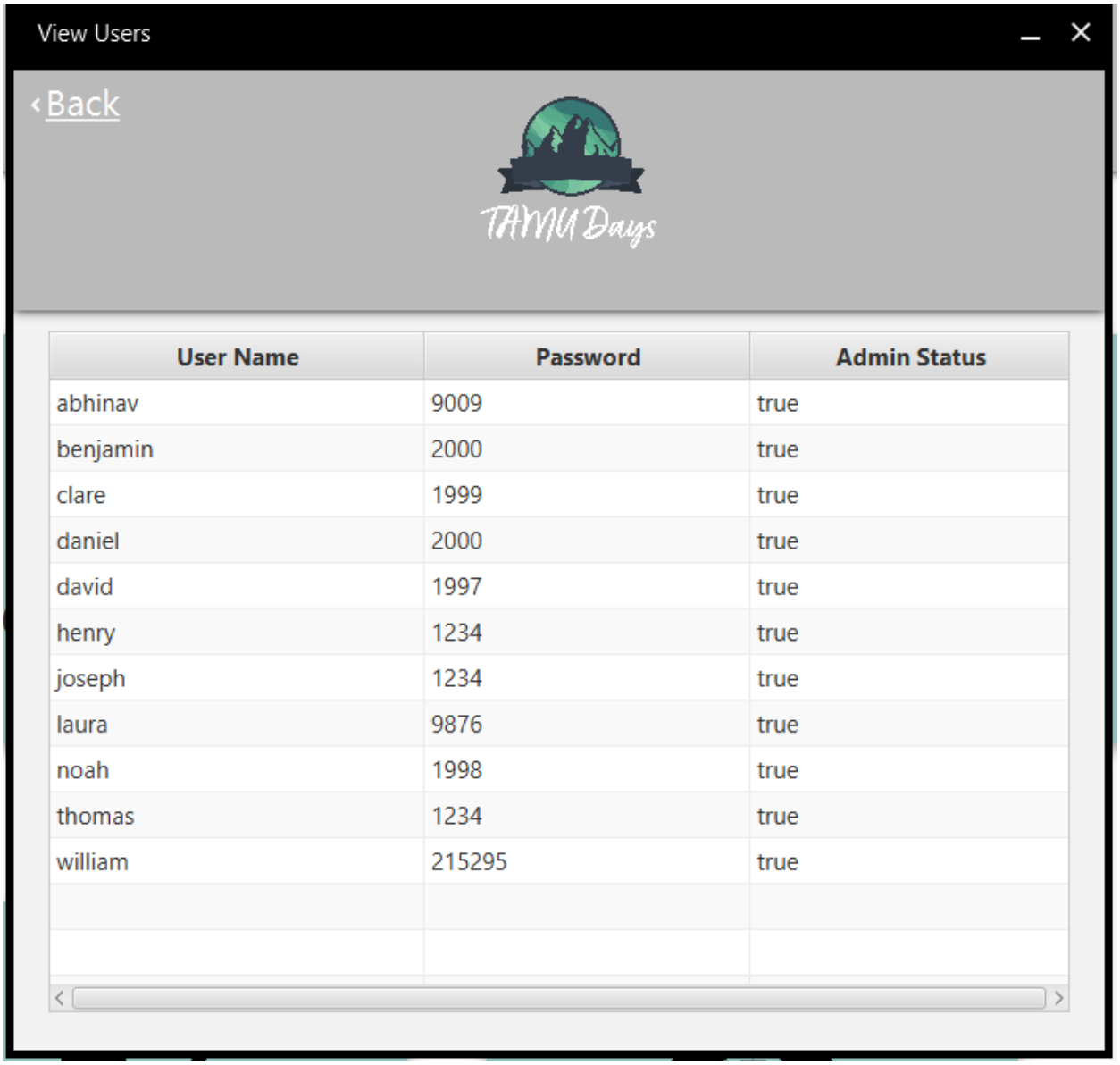
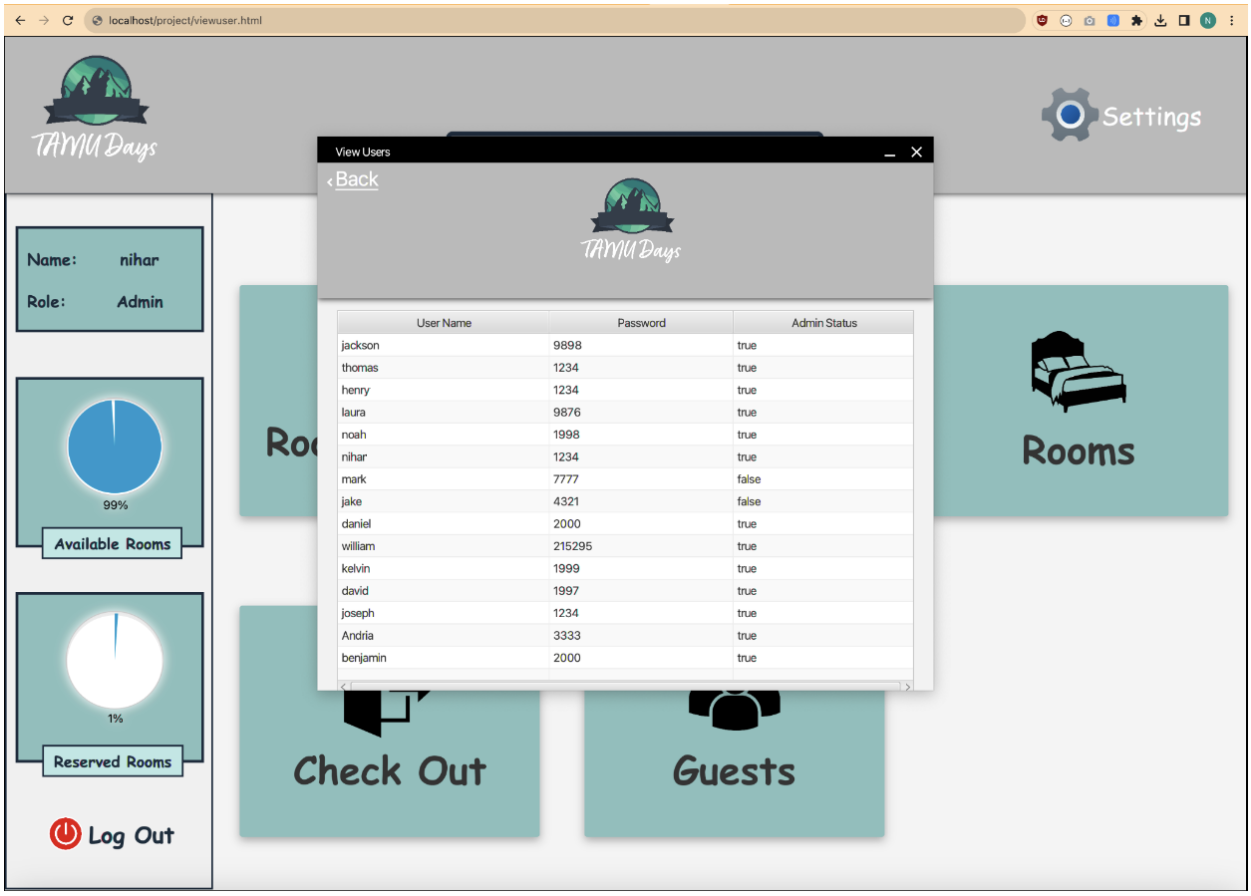


Figure 22. View all users

UI Sketch for Web Based Client:



Flg 23. View users

Use Case 8 : Administrator settings - Add

Actor: Admin

Goal: To add new user in the system

Preconditions: Admin must be logged in and username should be unique

Summary: This functionality is used to add other users by the user. Logged in user enters a valid username and password and clicks on the ADD button to register a new user. Here, the username can be alphanumeric but the password needs to be an integer. After a new user has been successfully registered, admin can check in the View All Users section to see if that user has been added or not as seen in figures.

Steps:

User Actions	System Responses
1. User enters a valid username and password	
2. User clicks “Add”	3. The system registers the new user in the redis.
4. User enters a username that is already registered	5. System prompts an error message that states “Username is already taken. Please enter another username”.
6. User clicks “Add” without entering username and password	7. System prompts an error message stating “Please fill all the fields”.

UI Sketch for Desktop Based Client:

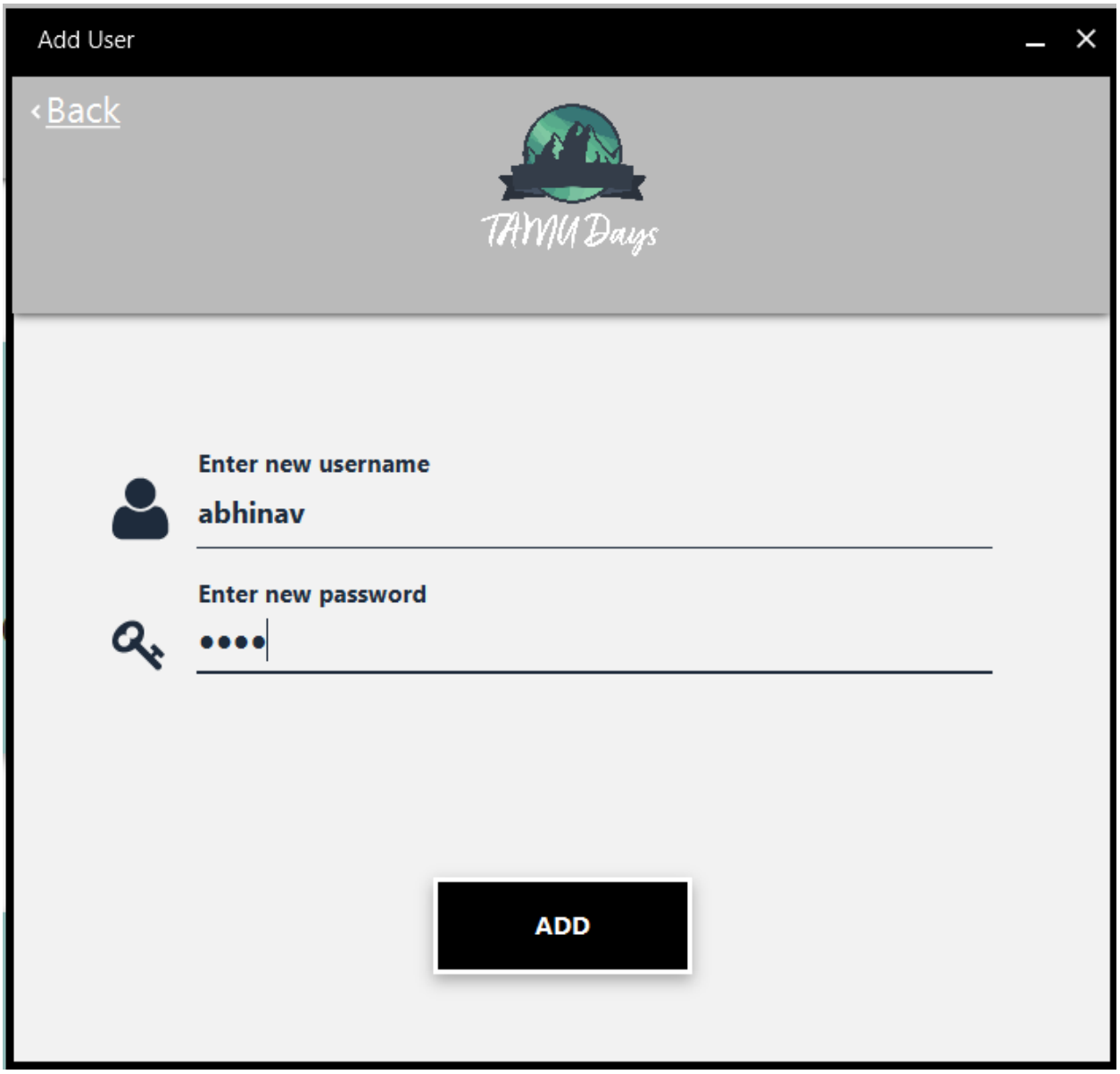


Figure 24. Add New User Page

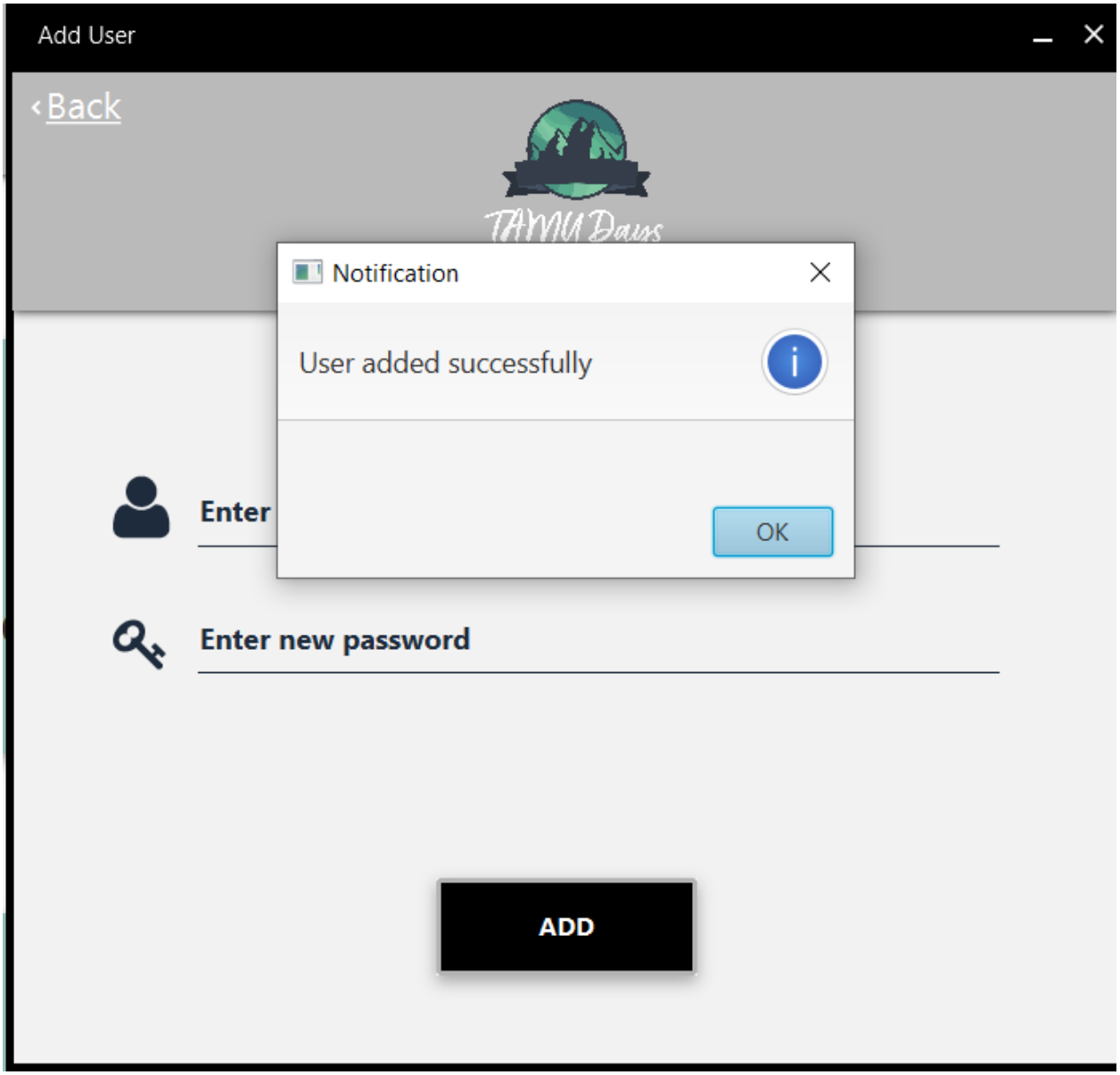


Figure 25. New User Added Successfully

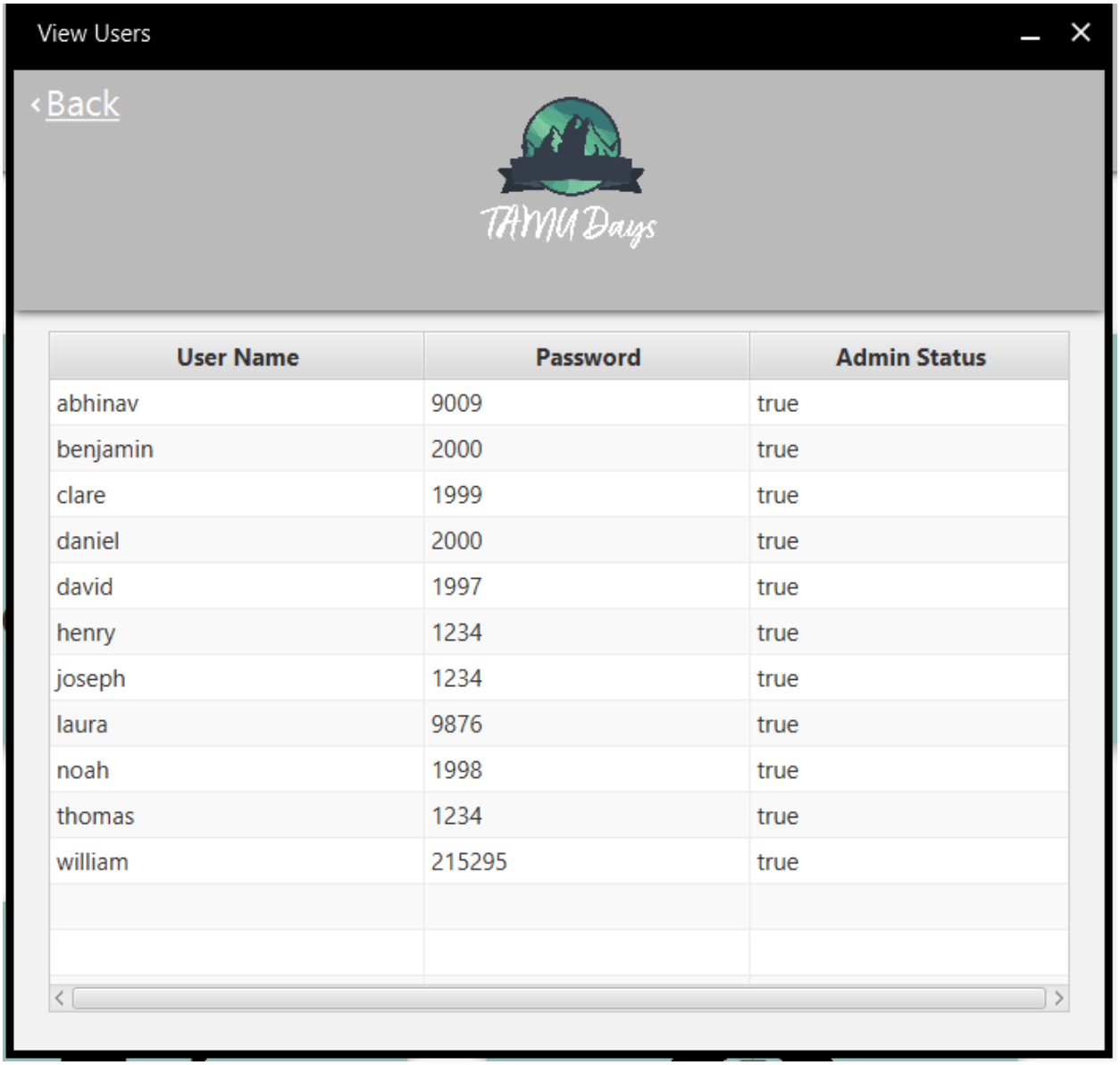


Figure 26. View Added User “abhinav”

UI Sketch for Web Based Client:

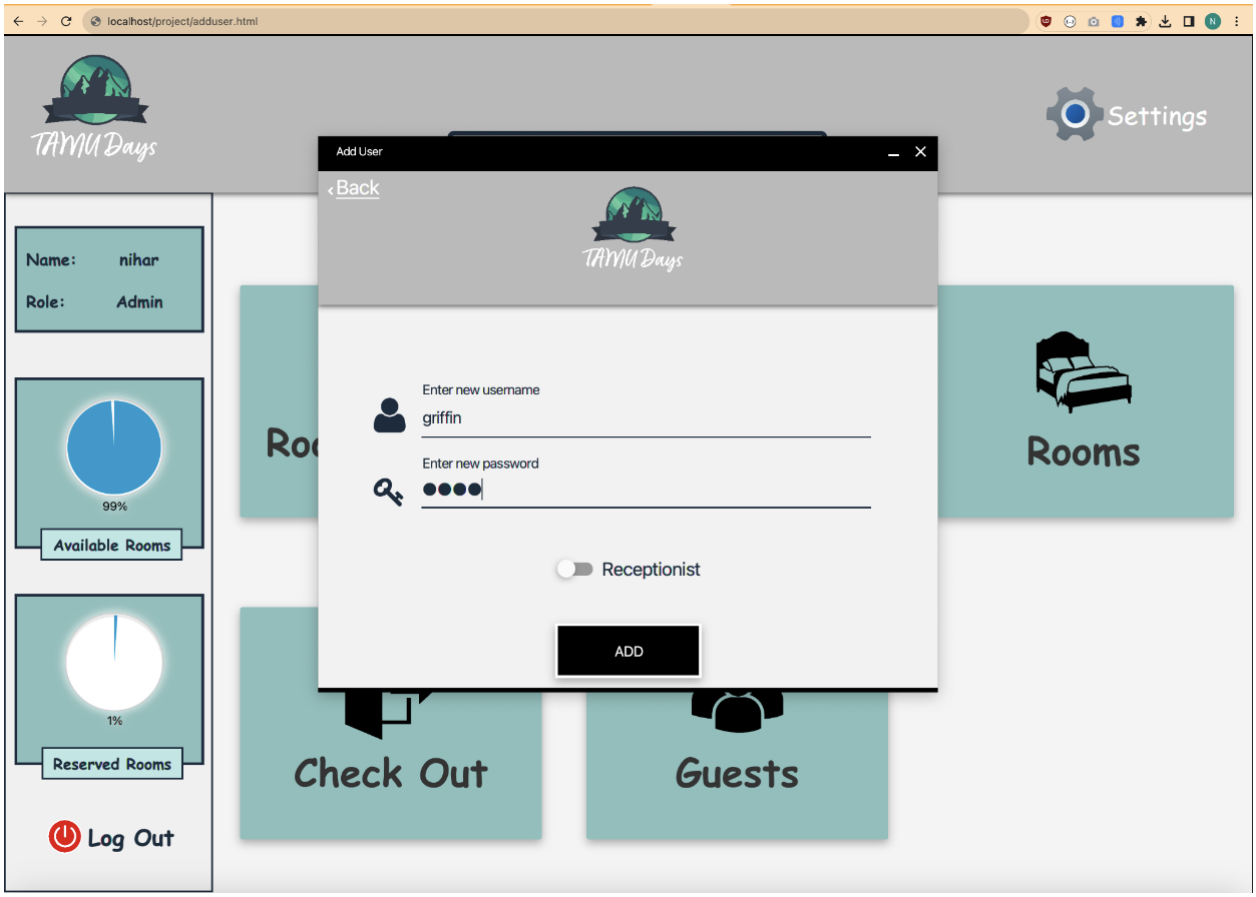


Fig 27. Add users

Use Case 9 : Administrator settings - Delete

Actor: Admin

Goal: To delete an existing user from the system

Preconditions: The user to be deleted must be present in the system.

Summary: This functionality is used to delete other admins by the user. Logged in user enters a valid username and clicks on the DELETE button to delete an already registered user. After a user has been successfully deleted, admin can check in the View All Users section to see if that user has been deleted as seen in the figure.

Steps:

User Actions	System Responses
1. User enters a valid username and password	
2. User clicks “Delete”	3. The system deletes the user from the redis.
4. User enters a username that is already presently in use.	5. System prompts an error message that states “You cannot delete yourself”.
6. User clicks “Delete” without entering username	7. System prompts an error message stating “Please fill all the fields”.

UI Sketch for Desktop Based Client:

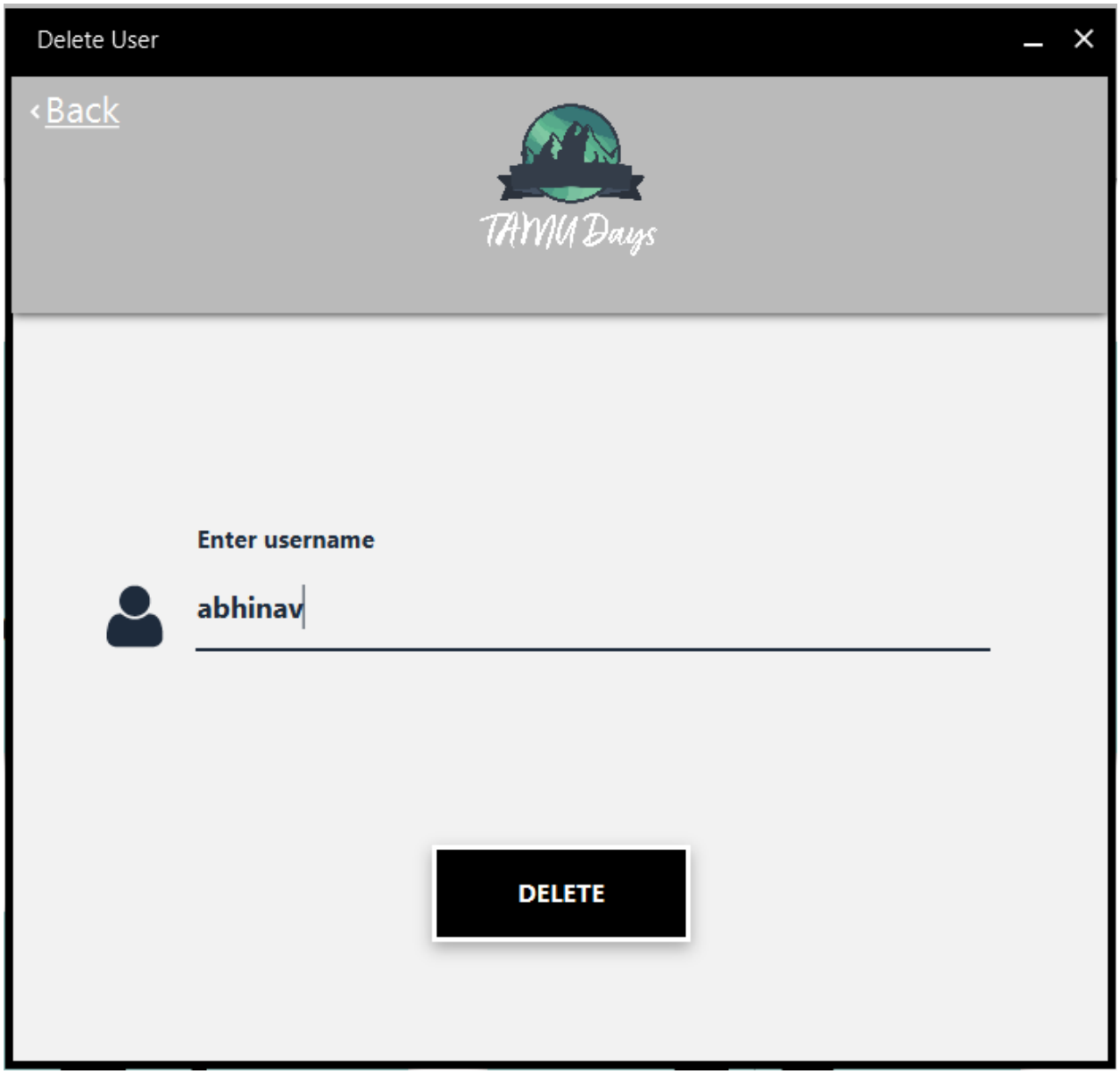


Figure 28. Delete User Page

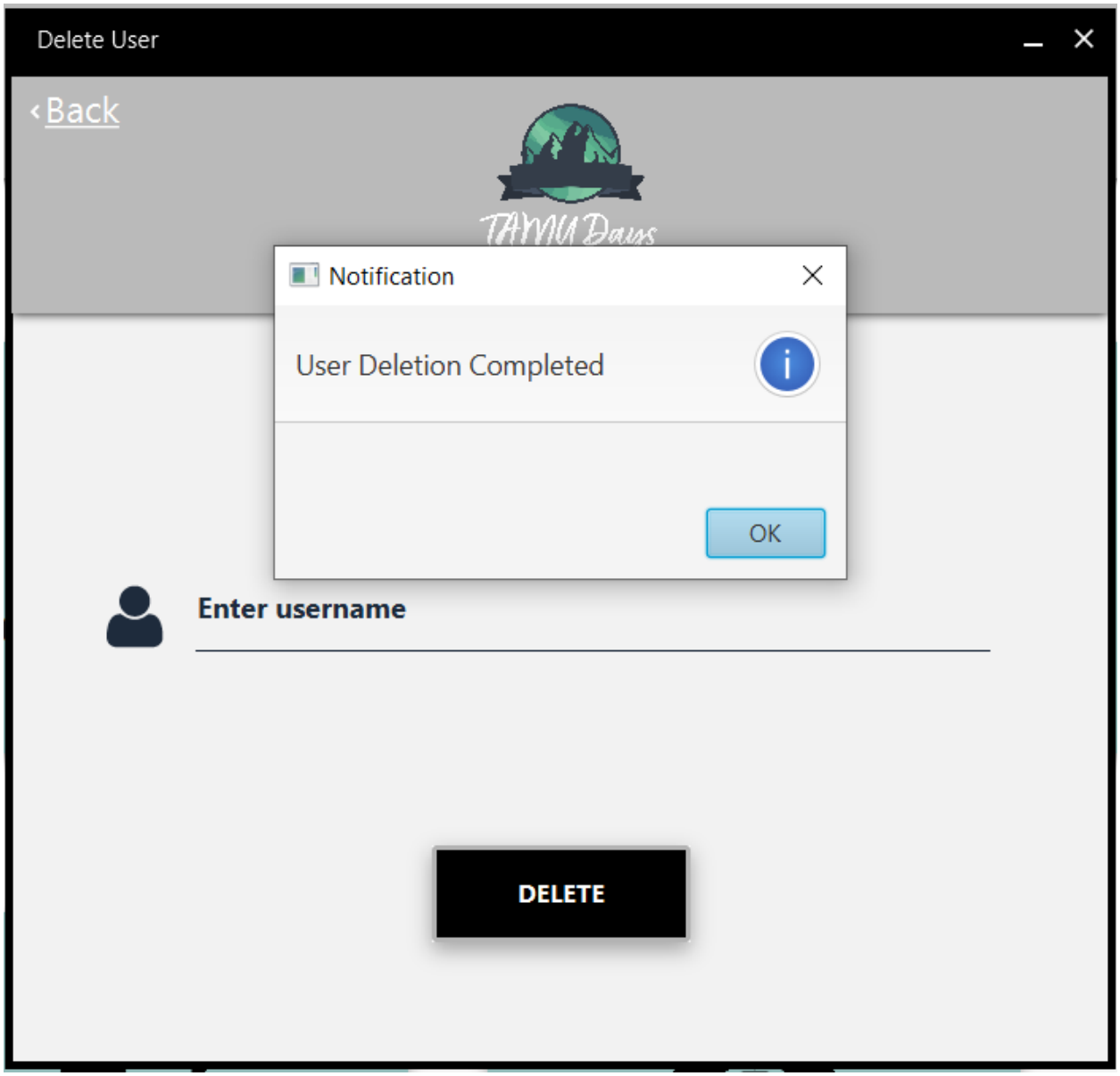


Figure 29. User Deleted Successfully

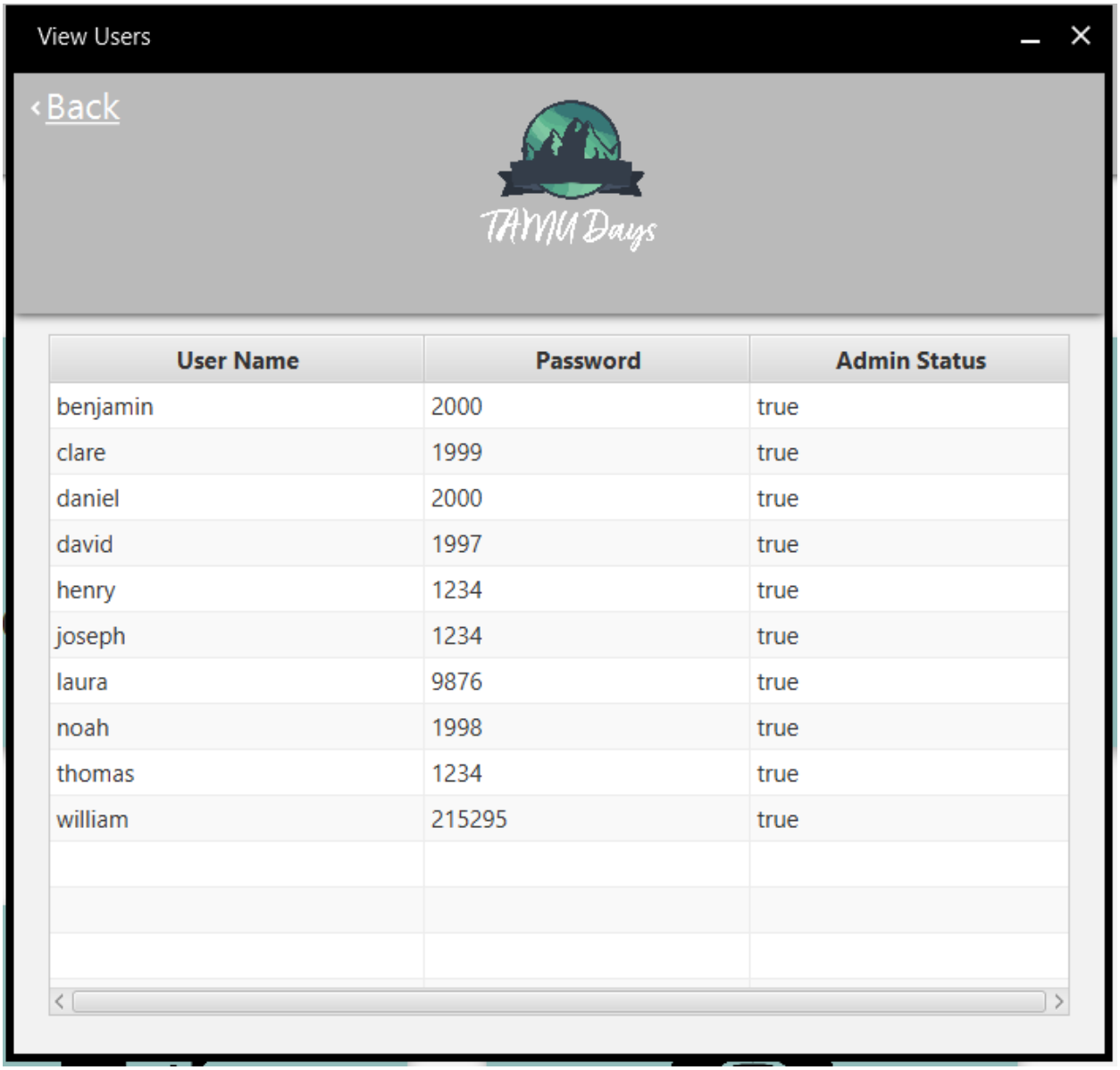


Figure 30. View Deleted User “abhinav”

UI Sketch for Web Based Client:

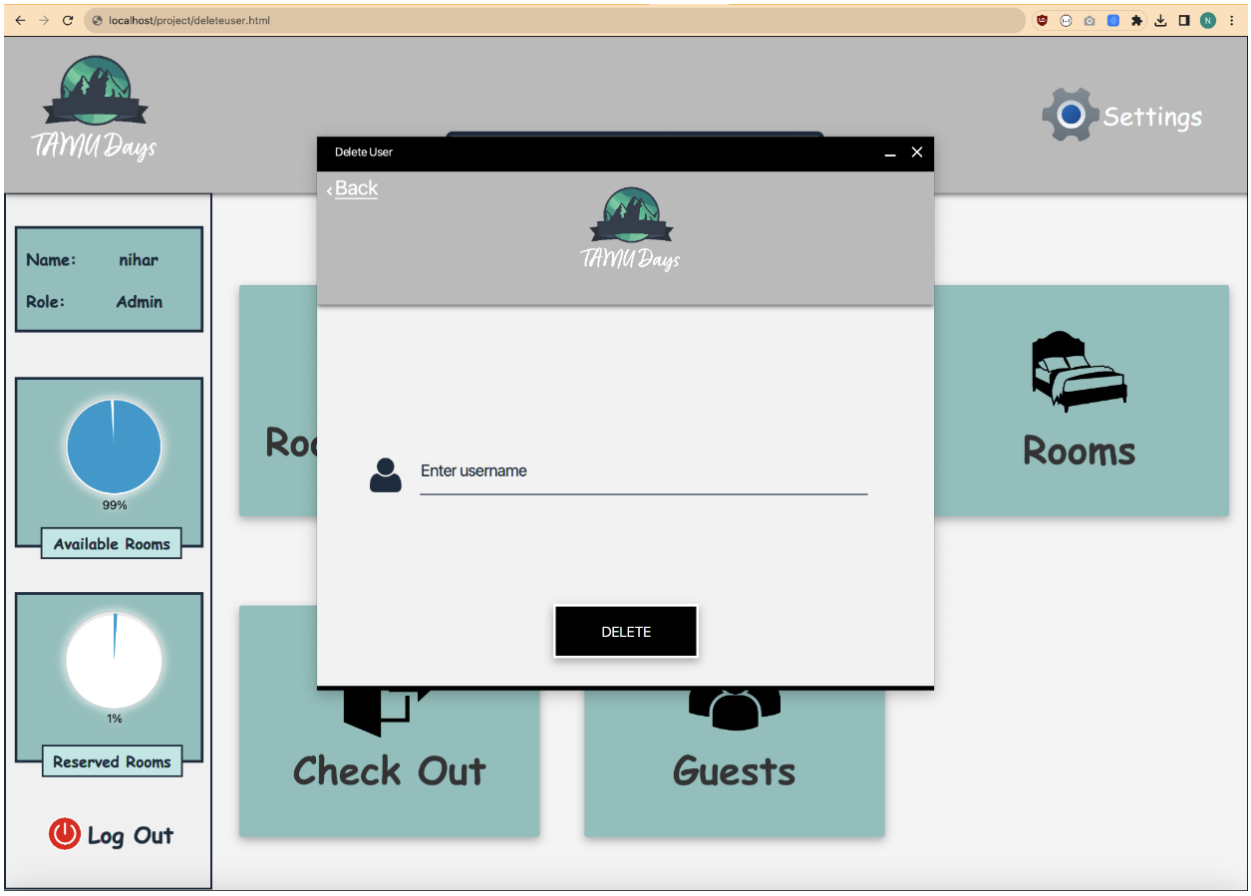


Fig 31. Delete user

Use Case 10 : Logout user

Actor: Receptionist/User, Admin

Goal: The user wants to log out of the system.

Preconditions: User must be logged into the system.

Summary: The Logout functionality is used to logout a logged in user from the system. A user is logged out from the system as soon as he/she presses the Log Out button present in the home page and a login screen reappears for the user.

Steps:

User Actions	System Responses
1. User clicks “Log Out”	2. System calls the logout function that redirects user to the login screen.

UI Sketch for Desktop Based Client:

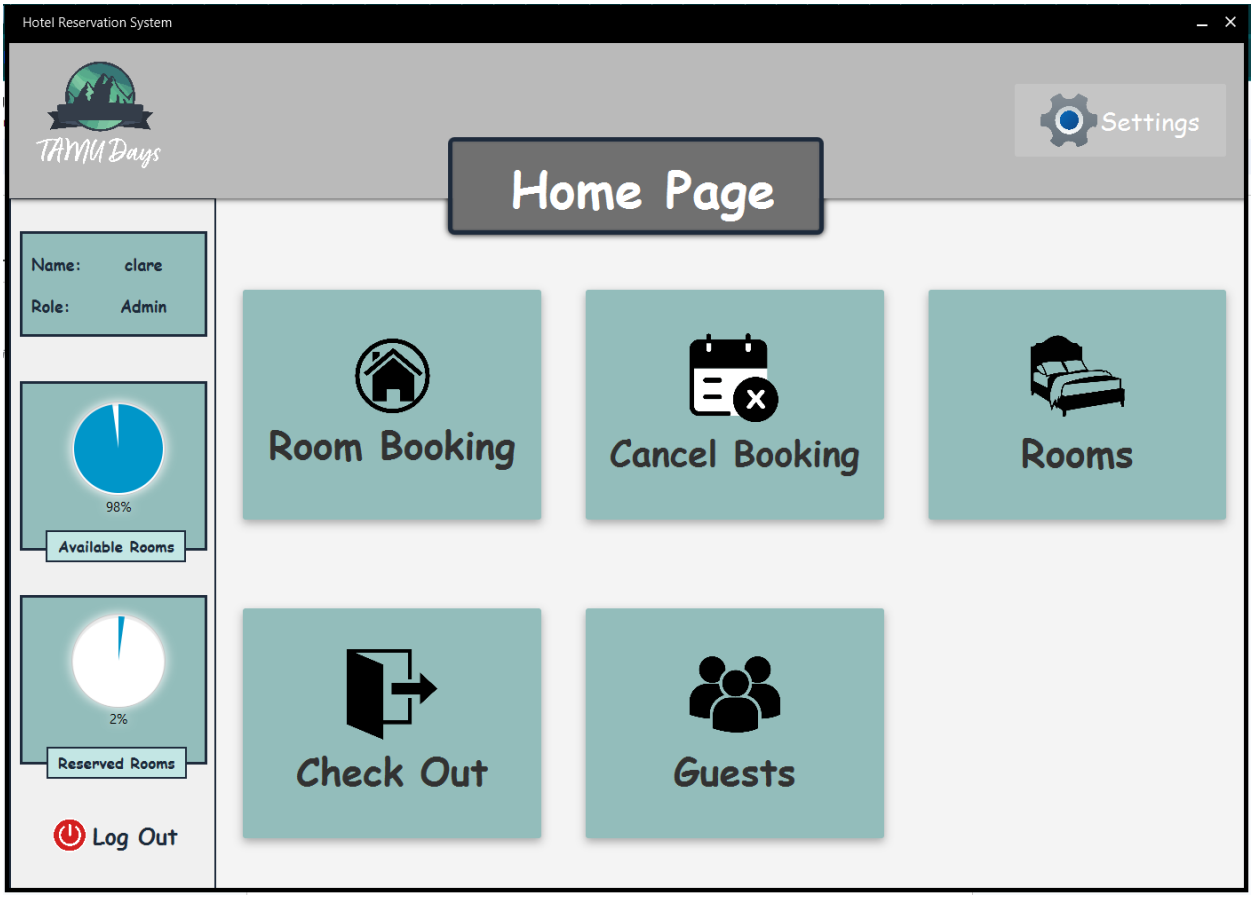


Figure 32. Logout Screen (Log Out button at the bottom left)

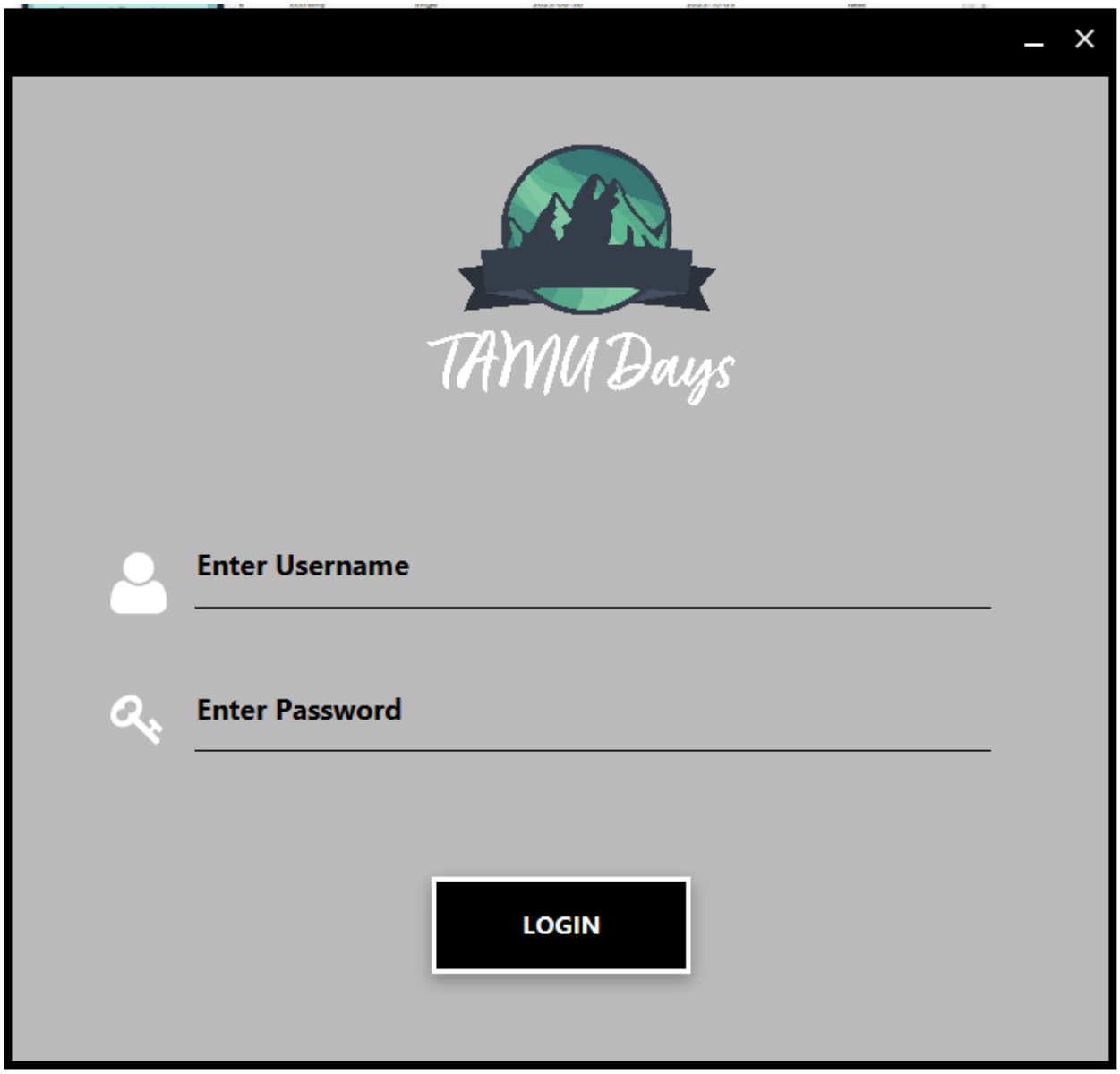


Figure 33. Logout Success Screen

UI Sketch for Web Based Client:

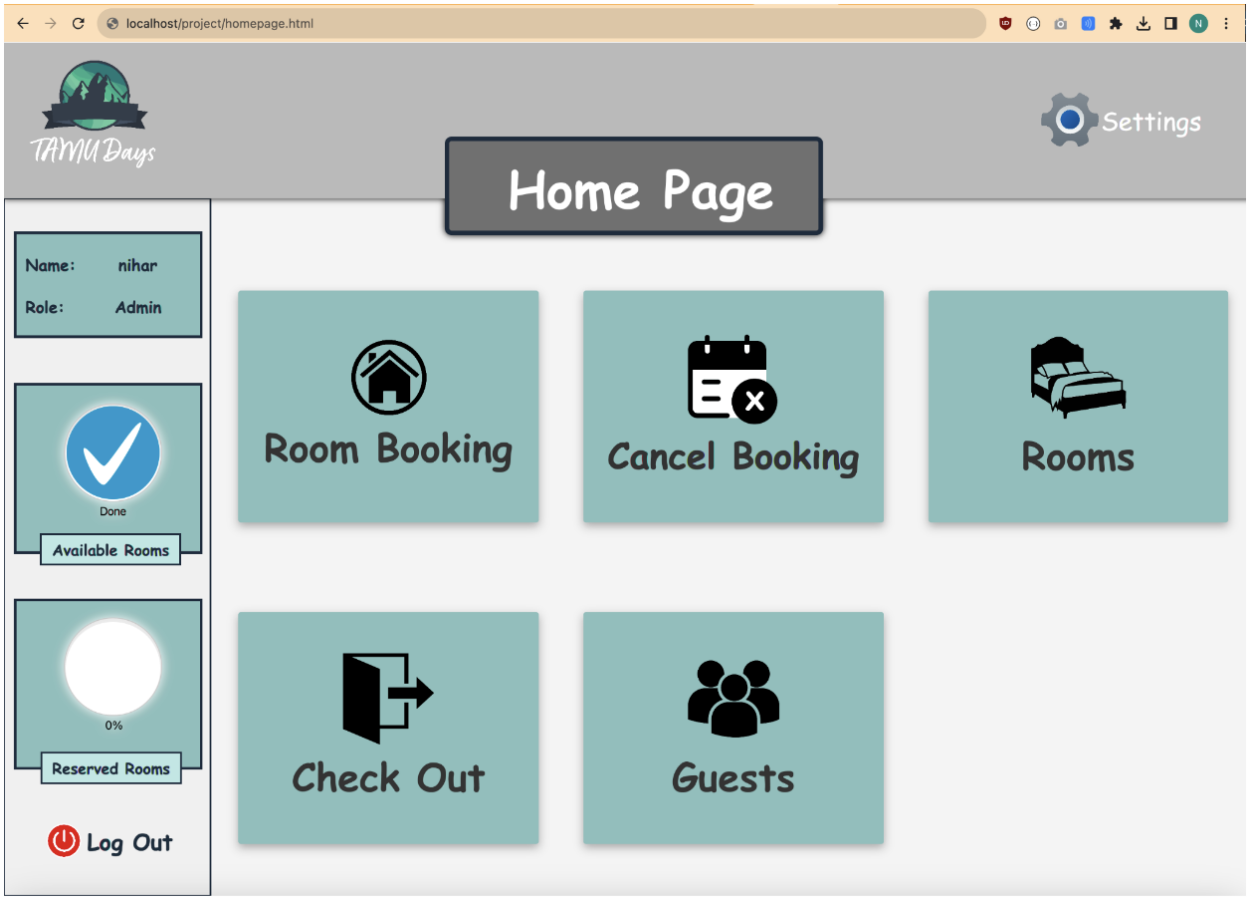


Fig 34. Logout Screen (Log Out button at the bottom left)

2. Database design:
Description of data entities and relationships, entity-relationship diagram, sample data.

The title of our project is “Hotel Reservation System”. We have developed a desktop application as well as web-based application for managing hotel reservations for the customers and the administrator.

Technologies used:

- JavaFX: Java-based framework for building graphical user interfaces (UIs) for desktop applications.
- F-XML: User Interface Markup Language.
- Java: Backend programming language for developing business logics.
- HTML: Standard Markup Language for documents.
- CSS: Describes the presentation of documents in HTML and XML.
- Redis: Open source, in memory data structure store used as a database.
- Apache Server: Open source cross platform web server software.
- MongoDB: Source available cross platform document oriented database.
- MySQL: Relational Database Management System.

Our project aims to create an efficient and reliable hotel reservation system which can be used by the hotel management.

The admins can:

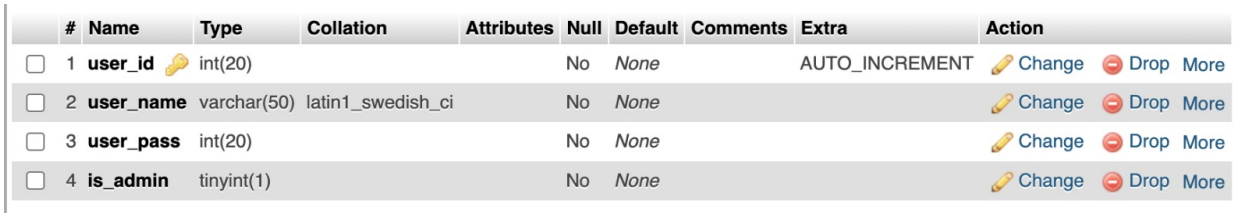
- check in customers if they wish to walk in directly or call up the hotel service contact.
- check out the customers at any interval within their fixed booking period in case of an early departure.
- add other administrators and receptionists, delete from the list of administrators and receptionists and check the booking details of a customer.
- check the status of all rooms of the hotel if they are occupied or available.

The project has the following three entities:

- USER

The user entity comprises the types of users for the project. The user_id serves as the primary key and uniquely identifies the user in the database. The user entity has the following attributes and their datatype:

Attributes	Datatype
user_id (Primary Key)	integer
user_name	varchar
user_pass	integer
is_admin	tinyint

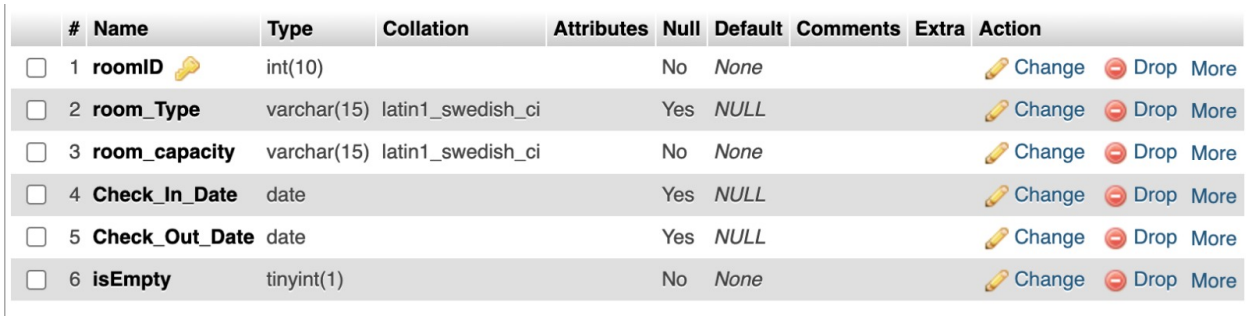


#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	user_id	int(20)			No	None		AUTO_INCREMENT	Change Drop More
2	user_name	varchar(50)	latin1_swedish_ci		No	None			Change Drop More
3	user_pass	int(20)			No	None			Change Drop More
4	is_admin	tinyint(1)			No	None			Change Drop More

- ROOM

The room entity comprises the details about the hotel rooms. The room entity is crucial in storing the reservation details for the tenure of the customer's stay. The roomID serves as the primary key in this entity and is used to uniquely identify the entity in the database. The room entity has the following attributes and their datatype:

Attributes	Datatype
roomID (Primary Key)	integer
room_Type	varchar
room_capacity	varchar
Check_In_Date	date
Check_Out_Date	date
isEmpty	tinyint



#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	**roomID**	int(10)			No	*None*			Change Drop More
2	**room_Type**	varchar(15)	latin1_swedish_ci		Yes	*NULL*			Change Drop More
3	**room_capacity**	varchar(15)	latin1_swedish_ci		No	*None*			Change Drop More
4	**Check_In_Date**	date			Yes	*NULL*			Change Drop More
5	**Check_Out_Date**	date			Yes	*NULL*			Change Drop More
6	**isEmpty**	tinyint(1)			No	*None*			Change Drop More

- GUEST

The guest entity comprises the details of the guest as well as room bookings made by the customers. The guest entity takes the roomID from the room entity as the foreign key to reference the guest info with the room booking and provide the same room id for both entities. The passport_number of the guest serves as the primary key for the guest entity and uniquely identifies it in the database. The guest entity has the following attributes and their datatype:

Attributes	Datatype
roomID (Foreign Key)	integer
Name	varchar
email	varchar
address	varchar
city	varchar
nationality	varchar
passport_number (Primary Key)	varchar
phoneNo	varchar
card_number	varchar
card_pass	varchar
number_of_days	integer
fees	double

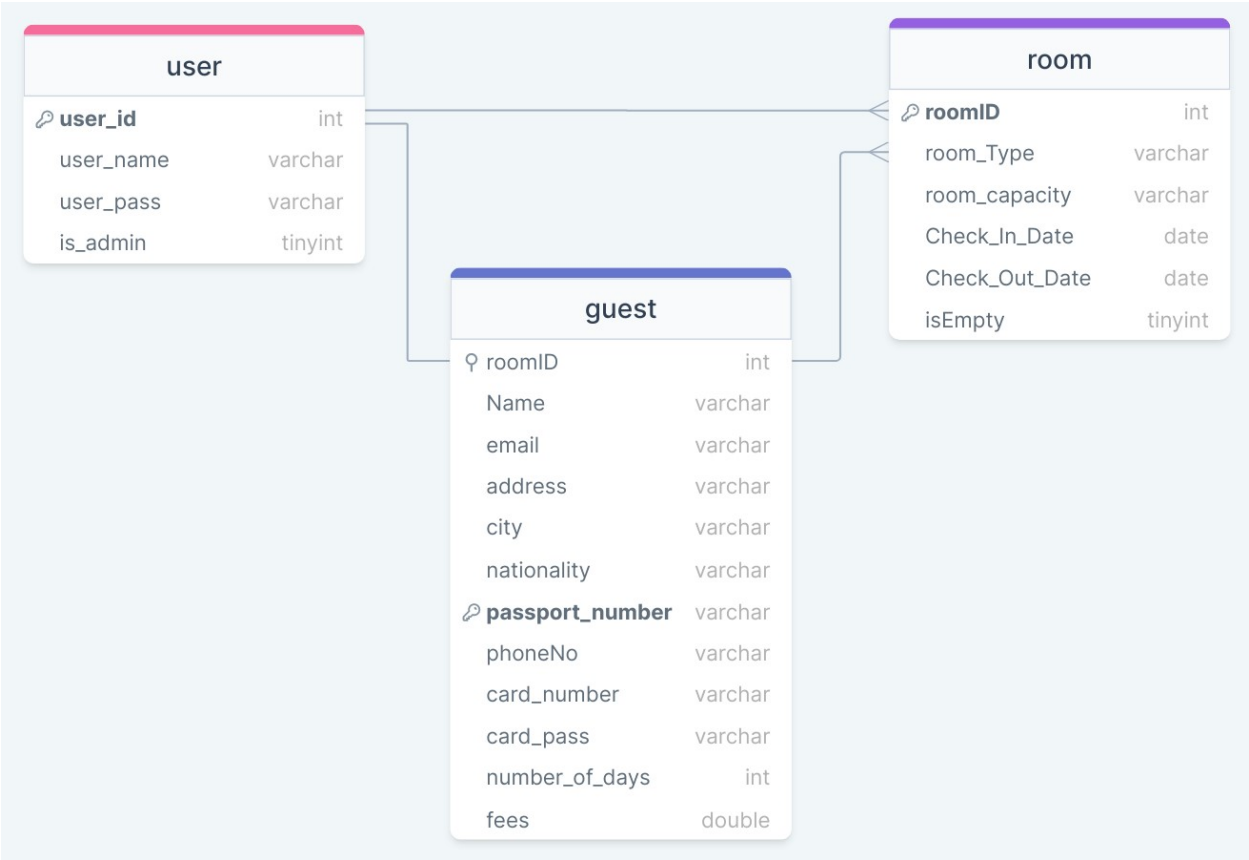
#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	**roomID**	int(10)			Yes	*NULL*			Change Drop More
2	**Name**	varchar(50)	latin1_swedish_ci		No	*None*			Change Drop More
3	**email**	varchar(50)	latin1_swedish_ci		Yes	*NULL*			Change Drop More
4	**address**	varchar(50)	latin1_swedish_ci		Yes	*NULL*			Change Drop More
5	**city**	varchar(50)	latin1_swedish_ci		No	*None*			Change Drop More
6	**nationality**	varchar(50)	latin1_swedish_ci		Yes	*NULL*			Change Drop More
7	**passport_number**	varchar(50)	latin1_swedish_ci		No	*None*			Change Drop More
8	**phoneNo**	varchar(50)	latin1_swedish_ci		Yes	*NULL*			Change Drop More
9	**card_number**	varchar(50)	latin1_swedish_ci		Yes	*NULL*			Change Drop More
10	**card_pass**	varchar(50)	latin1_swedish_ci		Yes	*NULL*			Change Drop More
11	**number_of_days**	int(10)			Yes	*NULL*			Change Drop More
12	**fees**	double			No	*None*			Change Drop More

The relationships between the entities of the project are as follows:

Entity 1	Relationship Type	Entity 2
USER	One-to-Many	ROOM
USER	One-to-One	GUEST
ROOM	Many-to-One	GUEST

Here a single user can book multiple rooms but once a room has been booked it is marked unavailable and hence cannot be booked by another user. So, the user to room is a one-to-many relationship.
A user will have only one guest instance at any time as the passport_number of each user is unique and can have only a one-to-one relationship with the guest entity.
Multiple rooms can be booked by a single guest over a course of time if there are enough rooms available to book and so the relationship between room and guest entities is a many-to-one relationship.

Entity-Relationship Diagram (ER Diagram):



SQL Code to Design Database:

Design for USER Entity

```
CREATE Table User (
user_id integer (10) PRIMARY KEY,
user_name varchar (50),
user_pass integer (20),
is_admin tinyint (1)
);
```

Design for ROOM Entity

```
CREATE Table Room (
roomID integer (10) PRIMARY KEY,
room_Type varchar (15),
room_capacity varchar (15),
Check_In_Date date,
Check_Out_Date date,
isEmpty tinyint (1)
);
```

Design for GUEST Entity

```
CREATE Table Guest (
roomID integer (10),
Name varchar (50),
email varchar (50),
address varchar (50),
city varchar (50),
nationality varchar (50),
passport_number varchar (50) PRIMARY KEY,
phoneNo varchar (50),
card_number varchar (50),
card_pass varchar (50),
number_of_days integer (10),
fees double,
FOREIGN KEY (roomID) REFERENCES Room (roomID)
);
```

Sample Data for Databases:

Data Entries for User Entity

INSERT INTO User (user_id, user_name, user_pass, is_admin) VALUES (1, “clare”, 1999, 1);

INSERT INTO User (user_id, user_name, user_pass, is_admin) VALUES (2, “luther”, 215295, 0);

INSERT INTO User (user_id, user_name, user_pass, is_admin) VALUES (3, “john”, 1997, 1);

INSERT INTO User (user_id, user_name, user_pass, is_admin) VALUES (4, “victor”, 1999, 1);

INSERT INTO User (user_id, user_name, user_pass, is_admin) VALUES (5, “thomas”, 1234, 0);

←T→	user_id	user_name	user_pass	is_admin
Edit Copy Delete	1	clare	1999	1
Edit Copy Delete	2	luther	215295	0
Edit Copy Delete	3	john	1997	1
Edit Copy Delete	4	victor	1999	1
Edit Copy Delete	5	thomas	1234	0

Data Entries for Room Entity

INSERT INTO Room (roomID, room_Type, room_capacity, Check_In_Date, Check_Out_Date, isEmpty) VALUES (1, “Economy”, “Single”, “2019-05-12”, “2048-05-20”, 0);

INSERT INTO Room (roomID, room_Type, room_capacity, Check_In_Date, Check_Out_Date, isEmpty) VALUES (11, “Economy”, “Double”, “2019-04-23”, “2048-04-15”, 1);

INSERT INTO Room (roomID, room_Type, room_capacity, Check_In_Date, Check_Out_Date, isEmpty) VALUES (21, “Economy”, “Triple”, “2019-04-16”, “2019-04-21”, 1);

INSERT INTO Room (roomID, room_Type, room_capacity, Check_In_Date, Check_Out_Date, isEmpty) VALUES (31, “Normal”, “Single”, “2019-04-14”, “2019-04-14”, 1);

INSERT INTO Room (roomID, room_Type, room_capacity, Check_In_Date, Check_Out_Date, isEmpty) VALUES (41, “Normal”, “Double”, “2023-10-05”, “2023-09-30”, 1);

INSERT INTO Room (roomID, room_Type, room_capacity, Check_In_Date, Check_Out_Date, isEmpty) VALUES (51, “Normal”, “Triple”, “2019-04-23”, “2019-04-15”, 1);

INSERT INTO Room (roomID, room_Type, room_capacity, Check_In_Date, Check_Out_Date, isEmpty) VALUES (61, “Vip”, “Single”, “2019-05-27”, “2019-05-04”, 1);

INSERT INTO Room (roomID, room_Type, room_capacity, Check_In_Date, Check_Out_Date, isEmpty) VALUES (71, “Vip”, “Double”, “2019-04-24”, “2019-04-30”, 0);

INSERT INTO Room (roomID, room_Type, room_capacity, Check_In_Date, Check_Out_Date, isEmpty) VALUES (81, “Vip”, “Triple”, “2019-05-16”, “2019-05-22”, 0);

←T→	roomID	room_Type	room_capacity	Check_In_Date	Check_Out_Date	isEmpty
Edit Copy Delete	1	Economy	Single	2019-05-12	2048-05-20	0
Edit Copy Delete	11	Economy	Double	2019-04-23	2019-04-15	1
Edit Copy Delete	21	Economy	Triple	2019-04-16	2019-04-21	1
Edit Copy Delete	31	Normal	Single	2019-04-14	2019-04-14	1
Edit Copy Delete	41	Normal	Double	2023-10-05	2023-09-30	1
Edit Copy Delete	51	Normal	Triple	2019-04-23	2019-04-15	1
Edit Copy Delete	61	Vip	Single	2019-05-27	2019-05-04	1
Edit Copy Delete	71	Vip	Double	2019-04-24	2019-04-30	0
Edit Copy Delete	81	Vip	Triple	2019-05-16	2019-05-22	0

Data Entries for Guest Entity

INSERT INTO Guest (roomID, Name, email, address, city, nationality, passport_number, phoneNo, card_number, card_pass, number_of_days, fees) VALUES (8, ”Herth”, “herth@gmail.com”, “London apts”, ”New Jersey”, “American”, “1452637818”, “9817263718”, “34567890”, “123”, 3, 600);

INSERT INTO Guest (roomID, Name, email, address, city, nationality, passport_number, phoneNo, card_number, card_pass, number_of_days, fees) VALUES (41, ”Penberthy”, “pen@gmai.com”, “Gateway Apts”, “Columbia”, “African”, “456789”, “615627891”, “23443234”, “123”, 9, 1800);

INSERT INTO Guest (roomID, Name, email, address, city, nationality, passport_number, phoneNo, card_number, card_pass, number_of_days, fees) VALUES (61, ”Jackson”, “jack@yahoo.com”, “Rapid Heights”, ”New York”, “China”, “87456789”, “8745678”, “8765456789”, “233”, 3, 900);

INSERT INTO Guest (roomID, Name, email, address, city, nationality, passport_number, phoneNo, card_number, card_pass, number_of_days, fees) VALUES (82, ”Smith”, “smith@gmail.com”, “United Buildings”, “Dallas”, “Indian”, “987653456”, “876545678”, “98765456789”, “232”, 6, 1200);

INSERT INTO Guest (roomID, Name, email, address, city, nationality, passport_number, phoneNo, card_number, card_pass, number_of_days, fees) VALUES (9, ”Flake”, “flake@hotmail.com”, “Queen streets”, “Apex”, ”Kenya”, “9876545678”, “9876545678”, “876545678”, “123”, 4, 800);

	roomID	Name	email	address	city	nationality	passport_number	phoneNo	card_number	card_pass	number_of_days	fees
Edit Copy Delete	8	Herth	herth@gmail.com	London apts	New Jersey	American	1452637818	9817263718	34567890	123	3	600
Edit Copy Delete	41	Penberthy	pen@gmail.com	Gateway Apts	Columbia	African	456789	615627891	23443234	123	9	1800
Edit Copy Delete	61	Jackson	jack@yahoo.com	Rapid Heights	New York	China	87456789	876545678	8765456789	233	3	900
Edit Copy Delete	82	Smith	smith@gmail.com	United Buildings	Dallas	Indian	9876543456	876545678	98765456789	232	6	1200
Edit Copy Delete	9	Flake	flake@hotmail.com	Queen streets	Apex	Kenya	9876545678	9876545678	876545678	123	4	800

Data Entries for Redis:

List of users registered in the application.

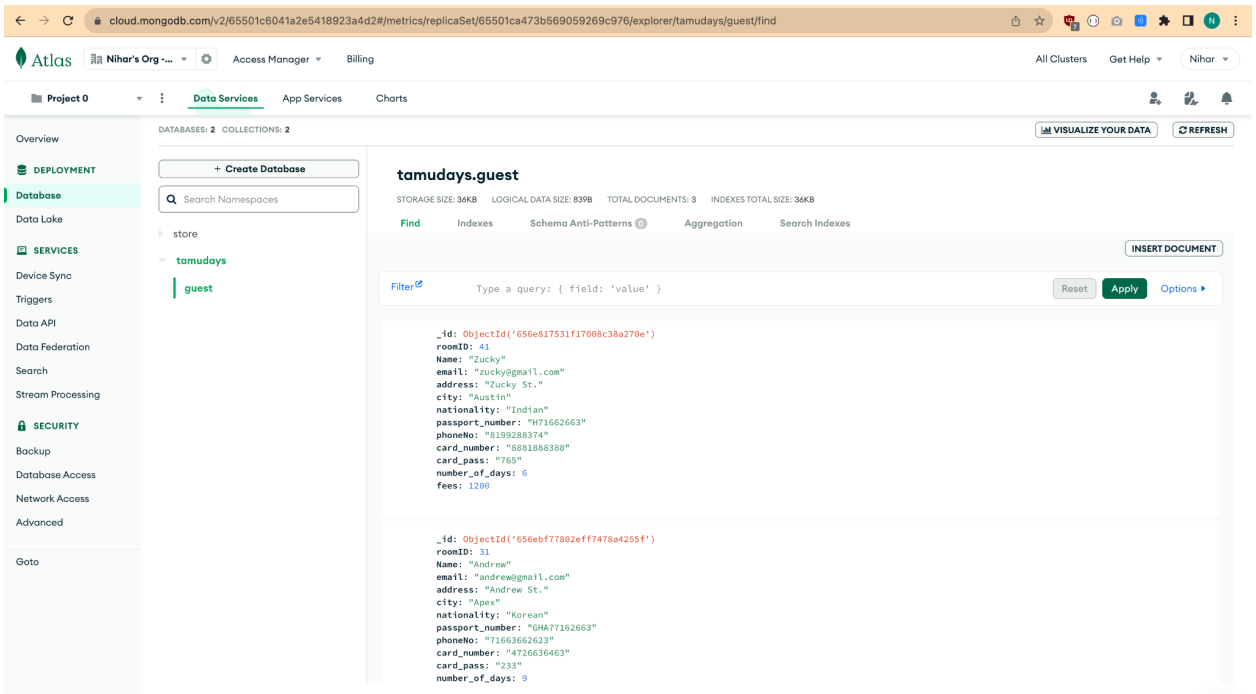
```
KEYS user:*

1) "user:joseph"
2) "user:kelvin"
3) "user:henry"
4) "user:thomas"
5) "user:Andria"
6) "user:daniel"
7) "user:william"
8) "user:benjamin"
9) "user:david"
10) "user:nihar"
11) "user:noah"
12) "user:jackson"
```

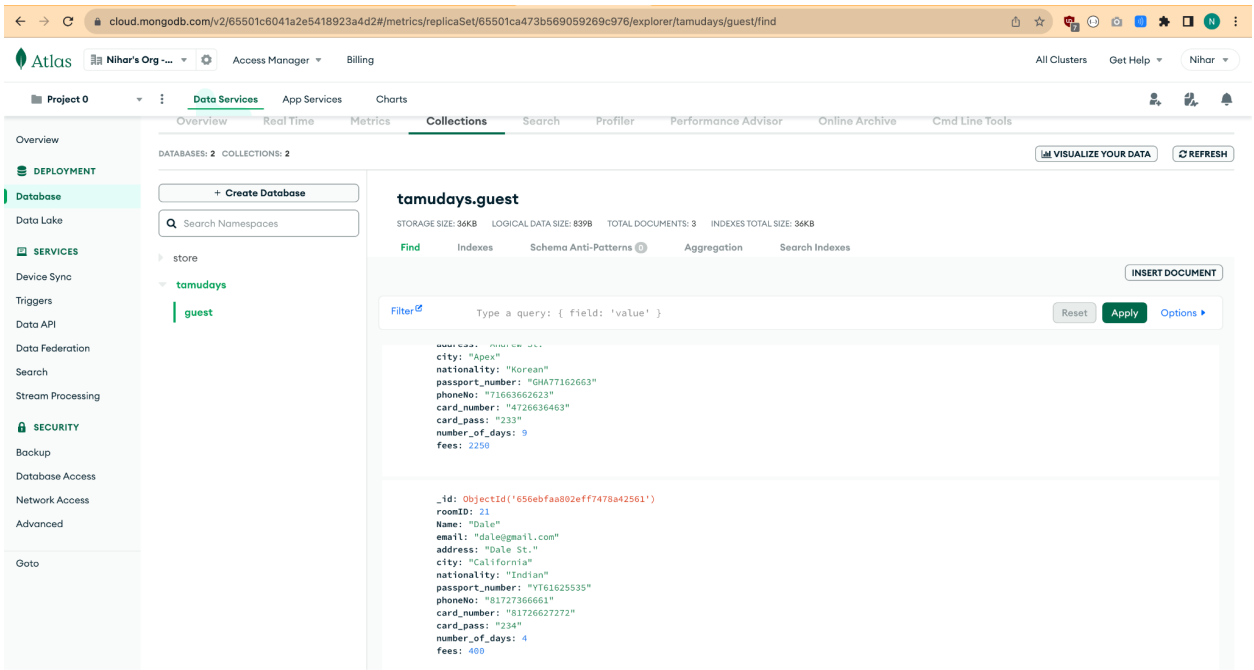
User’s login credentials.

```
HGETALL user:david

1) "user_pass"
2) "1997"
3) "is_admin"
4) "true"

HGETALL user:mark

1) "user_pass"
2) "7777"
3) "is_admin"
4) "false"

HGETALL user:joseph

1) "user_pass"
2) "1234"
3) "is_admin"
4) "true"
```

Data Entries for MongoDB:

List of guests staying in the hotel.




cloud.mongodb.com/v2/65501c6041a2e5418923a4d2#/metrics/replicaSet/65501ca473b569059269c976/explorer/tamudays/guest/find
Atlas
Nihar's Org -...
Access Manager
Billing
All Clusters
Get Help
Nihar
Project 0
Data Services
App Services
Charts
Overview
Real Time
Metrics
Collections
Search
Profiler
Performance Advisor
Online Archive
Cmd Line Tools
Overview
DEPLOYMENT
Database
Data Lake
SERVICES
Device Sync
Triggers
Data API
Data Federation
Search
Stream Processing
SECURITY
Backup
Database Access
Network Access
Advanced
Goto
DATABASES: 2 COLLECTIONS: 2
VISUALIZE YOUR DATA
REFRESH
+ Create Database
Search Namespaces
store
tamudays
guest
tamudays.guest
STORAGE SIZE: 36KB LOGICAL DATA SIZE: 839B TOTAL DOCUMENTS: 3 INDEXES TOTAL SIZE: 36KB
Find
Indexes
Schema Anti-Patterns
Aggregation
Search Indexes
INSERT DOCUMENT
Filter
Type a query: { field: 'value' }
Reset
Apply
Options
city: "Apex"
nationality: "Korean"
passport_number: "GHA77162663"
phoneNo: "71663662623"
card_number: "4726636463"
card_pass: "233"
number_of_days: 9
fees: 2250
_id: ObjectId('656ebfaa802eff7478a42561')
roomID: 21
Name: "Dale"
email: "dale@gmail.com"
address: "Dale St."
city: "California"
nationality: "Indian"
passport_number: "YT61625535"
phoneNo: "81727366661"
card_number: "81726627272"
card_pass: "234"
number_of_days: 4
fees: 400

3. Architectural design:
Description of client and server components; communication protocol (including data format), overall diagram of the system architecture.

In the context of client-server architecture, the terms "client" and "server" refer to distinct components that play different roles in the communication process.

Client Components:

A client is a piece of software or hardware that initiates a request to a server in a client-server architecture. Clients typically make requests to servers for data, services, or resources, and they wait for and process responses from the server.

Desktop Based Client:

- JavaFX is a framework for creating user interfaces in Java. It enabled us to design and customize UI elements, handle user interactions, and incorporate multimedia content.
- FXML is an XML-based language for defining the layout and structure of UIs in JavaFX. It provided a declarative way to define the user interface (UI) structure and layout. Together, we have used them to make up the client-side component responsible for creating the visual part of an application, allowing users to interact with it. In a client-server architecture, this client component communicates with the server component to request and exchange data.

Web Based Client:

- HTML serves as a fundamental building block for web-based clients. Used in conjunction with CSS for styling and JavaScript for interactivity, HTML enables the creation of visually appealing and dynamic web pages. It provides structure and content presentation, facilitating user interaction with web applications through browsers across various devices.
- Apache Server primarily acts as a web server, handling HTTP requests and serving static content like HTML, CSS, and JavaScript to clients. While Apache doesn't function as a web-based client itself, it plays a critical role in delivering web content to users.

Server Components:

A server is a computer program or device that provides resources, data, or services to clients in response to their requests. It listens for incoming requests from clients and processes these requests, providing the necessary resources or services.

- We have a socket-based server as a server component that uses network sockets to establish connections and exchange data with client applications. It listens for incoming connections, processes client requests, and sends responses. It's a fundamental part of client-server communication and enables data transfer over a network.
- We used microservices, as server components as they are independent, deployable units in a software architecture. They operate autonomously, communicate via REST APIs, and enable flexibility, scalability, and resilience. Each microservice, with specific business functionality, contributes to a modular, easily maintainable, and scalable system, fostering rapid development and continuous deployment. Each microservice uses an independent database.
  Room list uses MySQL database.
  Registered User list uses the Redis database.
  Guest list uses the MongoDB database.

Communication Protocol (including data format):

A communication protocol is a set of rules and conventions that govern the exchange of data between two or more parties in a communication system. These rules define how data is formatted, transmitted, received, and processed during communication.

HTTP protocol is used for communication. We have data exchange format as XML and we have used JavaFX (FXML) as the client component and a socket-based server as the server component, the communication protocol would involve the following:

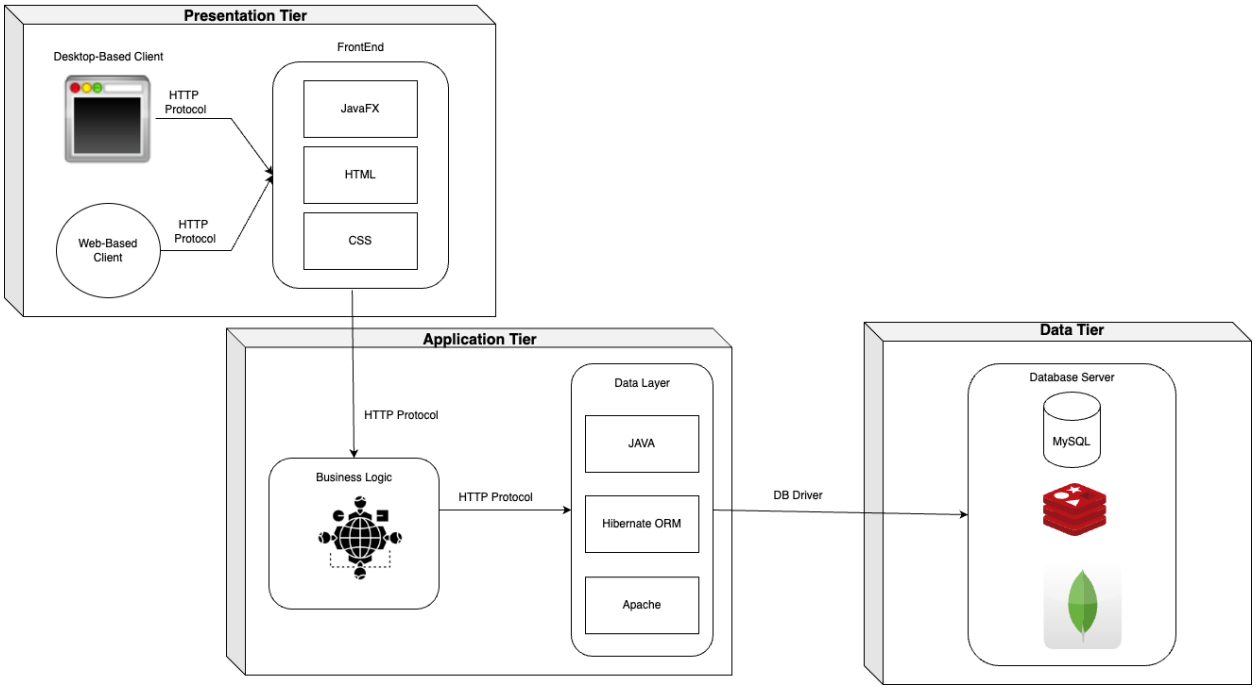
1. **HTTP:** It stands for Hypertext Transfer Protocol, is a fundamental protocol used for communication on the World Wide Web. It is an application layer protocol that facilitates the transfer of information between clients and servers. We have used below methods:
    - ***GET***: Retrieve a resource.
    - ***POST***: Submit data to be processed to a specified resource.

2. **XML Serialization:** Both the client and server need to agree on how data is serialized into XML format and deserialized back into usable data. XML provides a structured way to represent data, and this agreed-upon structure ensures that both ends can understand the content of XML messages.

3. **Message Structure:** Define a clear message structure for XML-based messages. This structure includes specifying how requests and responses are formatted as XML documents.

4. **Socket Communication:** The client initiates socket connections to the server, and the server listens for incoming connections. XML messages are sent over these socket connections as text-based data.

System Architecture



Components

The following section describes the main classes required for each component:

Presentation Tier:

The presentation tier entails all the pages from login, homepage, room booking, cancel booking, checkout, guests list, rooms list, admin control and logout.
The classes for this component are:

For desktop-based client:

- AddUser.fxml
- CancelBooking.fxml
- CancelBookingReceptionist.fxml
- CheckOut.fxml
- CheckOutReceptionist.fxml
- DeleteUser.fxml
- Guests.fxml
- GuestsReceptionist.fxml
- HomePage.fxml
- HomePageReceptionist.fxml

- ErrorPopUp.fxml
- Login.fxml
- ConfirmData.fxml
- RoomBooking.fxml
- RoomBookingReceptionist.fxml
- Rooms.fxml
- Users.fxml
- ViewUsers.fxml

For web-based client:

- adduser.html
- cancel.html
- checkout.html
- deleteuser.html
- guests.html
- homepage.html
- homepagereceptionist.html
- login.html
- confirmbooking.html
- roombooking.html
- rooms.html
- users.html
- viewusers.html

Application Tier:

The application tier is a crucial layer in a software architecture, residing between the presentation layer and data storage. It houses the business logic and processing, ensuring the functionality and performance of an application. This tier handles user requests, executes operations, and interfaces with databases to deliver dynamic and responsive applications.

The following files form the crux of the application tier. Each of the controller acts as a microservice as each of them has different functionalities and communicates via REST APIs:

- **AddUserController.java** - Add a new user as an admin/receptionist in the application.
- **CancelBookingController.java** - Cancels a booking of the guest.
- **CheckOutController.java** - Check Out a booking of the guest.
- **DeleteUserController.java** - Delete a user from the application.

- **GuestsController.java** - View the list of guests staying in the hotel.
- **HomePageController.java** - Display the home page for the application.
- **LoginController.java** - Displays the login page of the application.
- **ConfirmDataController.java** - Confirms the inputted data while booking a room for the guest.
- **RoomBookingController.java** - Books a room for the guest.
- **RoomsController.java** - View the list of all the rooms in the hotel.
- **UsersController.java** - Display a page to add, view and delete user in the application.
- **ViewUsersController.java** - View the list of users either admin or receptionist registered in the application.

Data Tier:

The data tier, integral to a software architecture, manages data storage and retrieval. It includes databases and storage systems, serving as the foundation for storing, organizing, and retrieving information. This tier ensures data integrity, security, and scalability, playing a key role in supporting the overall functionality and performance of an application.

The following databases form the foundation for the data tier:

- MySQL
- Redis
- MongoDB

4. A runnable system. You should personalize and polish the UI to improve its uniqueness and look-and-feel.

5. Video recordings of user acceptance tests. Each recording is for a use case.

(4th and 5th point stated above is covered in the below videos)

Usecase 1 - Login:
https://youtu.be/Lc424crkPzk

Usecase 2 - Room Booking:
https://www.youtube.com/watch?v=o5DkMbSEGek

Usecase 3 - Cancel Booking:
https://www.youtube.com/watch?v=qV2N_V4v9s0

Usecase 4 - Rooms:
https://www.youtube.com/watch?v=mmj8W6F_9n4

Usecase 5 - Guests:
https://www.youtube.com/watch?v=sYwrtup6EkM

Usecase 6 - Checkout:
https://www.youtube.com/watch?v=Jhk2-f6sXFk

Usecase 7 - View User:
https://www.youtube.com/watch?v=Y3yfWf3DcaY

Usecase 8 - Add User:
https://www.youtube.com/watch?v=9g8UOYrCHec

Usecase 9 - Delete User:
https://www.youtube.com/watch?v=nMbozer9sKQ

Usecase 10 - Logout:
https://www.youtube.com/watch?v=290yir7OqMw

Usecase 11 - (New usecase) Use application as Receptionist:
https://www.youtube.com/watch?v=w4e7jPAODnI

6. Manual for installation and usage with information on necessary libraries, frameworks, and running guidelines.

You may also refer this video for setup:
https://www.youtube.com/watch?v=8Q6tTWjVCbo

OR

To run this project, you need following softwares:

- **Apache Netbeans IDE 12.2** - It is an open-source integrated development environment (IDE) primarily focused on Java.

- **XAMPP** - It is a widely-used open-source software package that facilitates the setup and management of a local web development environment.

- **RedisInsight** - It is a graphical user interface (GUI) and management tool for Redis, an open-source, in-memory data structure store.

- **MongoDB Atlas** - It is designed to provide a simplified and efficient way to work with MongoDB databases in a cloud environment.

- **JDK 1.8** - It is a version of the Java Development Kit, which is a set of software development tools for developing Java applications.

- **jedis-5.1.0.jar** - If you are working with Redis in a Java application, you can use the Jedis library to connect to your Redis server, send commands, and process the results.

- **mongo-java-driver-3.11.2.jar** - If you are developing a Java application that needs to interact with a MongoDB database, you would use the MongoDB Java Driver. This includes tasks such as connecting to a MongoDB server, querying and updating data, and handling BSON documents.

- **charm-glisten-4.4.1-javadoc.jar** - This JAR file typically contains the JavaDoc documentation for the Charm Glisten library version 4.4.1. JavaDoc provides information about classes, methods, and other aspects of the library to help developers understand how to use it.

- **charm-glisten-4.4.1.jar** - This JAR file contains the compiled code for the Charm Glisten library version 4.4.1. Charm Glisten is a JavaFX library that provides additional UI components and styles for JavaFX applications.

- **jfoenix-8.0.8.jar** - JFoenix is a JavaFX material design library. It includes UI controls and styles that follow the material design guidelines. Developers can use JFoenix to enhance the visual appearance of their JavaFX applications.

- **mysql-connector-java-5.1.23-bin.jar** - This JAR file contains the MySQL Connector/J library version 5.1.23. Connector/J is a JDBC driver for MySQL databases, allowing Java applications to connect to and interact with MySQL databases.

- **mysql-connector-java-8.0.15.jar** - Similar to the previous MySQL Connector/J library, this JAR file contains version 8.0.15. It's a newer version, and developers may choose between versions based on their application's compatibility and requirements.

You can use this URI in your terminal to login to redis.

Cloud Redis URI:
redis://:YESAw1tH0qbEoJhDzNCzO1e84d0w6iAW@redis-12931.c52.us-east-1-4.ec2.cloud.redislabs.com:12931/0

Cloud Mongo URI:
mongodb+srv://nihar1999:As9uyZ~k@cluster0.3kkupx9.mongodb.net/?retryWrites=true&w=majority

Mongo Database name: tamudays
Mongo Collection name: guest

After having all the necessary softwares and libraries:

1. Open Netbeans IDE, and open project folder submitted in this assignment in the IDE.
2. Download provided JARs and import one by one by going to the *Libraries* folder in the *Projects* section present at the left side of the IDE.
3. Right click on the Libraries folder, click *Add Jar/Folder*, and choose the relevant JAR file from your directory and finally click on Choose button.
4. Do the above step one by one for all the necessary JAR files.
5. Ensure that JDK 1.8 is selected by the Netbeans. This can be checked by going to the *Java Platforms* in the *Tools* section present in the IDE's menu bar.
6. Add platform as JDK 1.8 and click on Close.
7. Open XAMPP and click on Start All to start all the servers.
8. Go to this link: http://localhost/phpmyadmin/index.php and create a new database named hotel.
9. Import the database hotel.sql attached in the zip for this assignment. This would import one table 'room' in the database.

After having the setup, it's time to run the application.

Click on the Run Project button (Green play button) located at the top of the IDE.

This would start the application and a login page would appear.

You may use login credentials such as:
For admin:
Username: jackson
Password: 9898

For Receptionist:
Username: mark
Password: 7777

You may now play around with the application similarly as we have shown in the video recordings.