



Department of Computer Science and Information Technology

Frostburg State University

Database Systems- COSC 641

Group Project: Hotel Management System

Team Members:

Name	Email
1. Navya Sree Sriram	nsriram0@frostburg.edu
2. Vageesh Hegde	vhegde0@frostburg.edu
3. Suresh B Vuppala	sbvuppala0@frostburg.edu

Under the guidance of

Dr. Nooh Bany Muhammad

2024

Applied Compute Science

Index:

1. Project Summary.....	4
2. Project Description.....	4
3. Problem Statement.....	5
4. Objectives.....	5
5. Software's Used.....	5
6. Database Schema.....	7
7. User Interface Design.....	17
8. Backend Implementation.....	43
9. Test Cases.....	46
10. Cloud Database.....	47
11. Conclusion and Future work.....	54

List of Figures

Figure number	Title of the figure	Page Number
6.1	ER Diagram	15
6.2	Relational Diagram	16
7.1	Use Case Diagram	17
7.2	Login Dashboard	18
7.3	Sign Up Dashboard	18
7.4	Navigation Section of Customer Dashboard	19
7.5	Webpage to Display Available Rooms	19
7.6	Webpage to show Booking History	30
7.7	Navigation Section of Staff Dashboard	31
7.8	Webpage to View all Bookings, checkin and checkout Date	31
7.9	Webpage to View all Services	32
7.10	Webpage to View all Reviews	32
7.11	Webpage to View all Amenities	33
7.12	Webpage to add Amenities to the List	33
7.13	Webpage to add Service to the List	34
7.14	Webpage to Search a Customer by Email	34
7.15	Overview of staff Dashboard	35
7.16	Graph to Show Rating and Payment Details	35
7.17	Graph to Show Room and Breakfast Details	36
7.18	Graph to Show Saty Duration and Booking Details	36
7.19	Graph to Show Reservation Trends	36
8.1	QR code sent to registered email	44

List of Tables

Table number	Title of the table	Page number
6.1	Database Schema	11
6.2	Views List	12
6.3	Sample Data for Customer Entity	13
6.4	Sample Data for Booking Entity	14
6.5	Sample Data for Receipt Entity	14
6.6	Sample Data for Room Entity	14
6.7	Sample Data for Service Entity	15
6.8	Sample Data for Staff Entity	15
6.9	Sample Data for Review Entity	17
6.10	Sample Data for Amenities Entity	18
6.11	Sample Data for RoomAmenities Entity	19
6.12	booking_people_counts - View	20
6.13	booking_stay_duration_counts - View	20
6.14	booking_summary - View	20
6.15	booking_year_counts - View	20
6.16	booking_year_month_counts - View	21
6.17	payment_counts - View	21
6.18	rating_counts - View	22
6.19	review_details - View	23
6.20	room_group_counts - View	23
6.21	service_details - View	24
8.1	Routes Table	46
9.1	Test Cases	47

Detailed Project Requirements:

Objectives:

Task	Status
1. Clearly define the purpose and scope of the database.	Completed
2. Identify the target users of the database.	Completed
3. Analyze the current database systems and compare with the proposed solution.	Completed
4. Develop a plan for future database maintenance and upgrades	Completed
5. Investigate advanced database concepts and technologies, such as NoSQL, cloud computing, and big data.	Completed

Data Requirements:

Task	Status
1. Identify the entities and relationships in the database.	Completed
2. Define the attributes for each entity.	Completed
3. Determine the data constraints, such as primary key, foreign key, and not null constraints.	Completed
4. Consider data storage and retrieval methods, such as indexing and partitioning.	Completed
5. Analyze the data quality and implement data cleaning techniques.	Completed
6. Consider the data privacy and security issues.	Completed
7. Investigate data warehousing and business intelligence concepts.	Completed

User Requirements:

Task	Status
-------------	---------------

1. Identify the user interactions with the database, such as insert, update, delete, and retrieve.	Completed
2. Determine the user views and reports required for data analysis.	Completed
3. Develop user authentication and authorization mechanisms.	Completed
4. Consider the user interface design and accessibility	Completed
5. Investigate the use of mobile devices and web services for accessing the database.	Completed

Technical Requirements:

Task	Status
1. Choose a database management system (DBMS) for the project.	Completed
2. Design the database schema using ER diagram.	Completed
3. Develop the database using the chosen DBMS.	Completed
4. Implement backup and recovery procedures.	Completed
5. Consider database scalability and performance optimization.	Completed
6. Investigate advanced database concepts, such as transactions, concurrency control, and recovery.	Completed
7. Consider the use of NoSQL, cloud computing, and big data technologies.	Completed

Testing:

Task	Status
-------------	---------------

1. Test the database for data integrity, consistency, and accuracy	Completed
2. Verify the user interactions and reports for correct results	Completed
3. Test the backup and recovery procedures.	Completed
4. Evaluate the performance and scalability of the database.	Completed
5. Evaluate the security and privacy of the database.	Completed

Documentation:

Task	Status
1. Provide a complete documentation of the database design, including ER diagram and data dictionary.	Completed
2. Provide a user manual for the database.	Completed
3. Document the testing results and performance analysis.	Completed
4. Provide a maintenance plan for the database.	Completed
5. Report on the advanced database concepts and technologies investigated.	Completed

1. Project Summary:

The hotel room booking system database project aims to develop a comprehensive solution for efficient room reservations. The database will serve as the core of the hotel's operations, facilitating

guest bookings, room management, and administrative tasks. The system will have guest profiles, room details, and reservation information. It will maintain detailed guest information, including personal details and contact information, to personalize their experience. The room inventory will be managed by room types, availability, and pricing, ensuring accurate booking and allocation. Reservation records are stored to track reservations, check-in/check-out dates, and optimizing room allocation and guest services.

2. Project Description:

Our database project is focused on developing a comprehensive Hotel Management System that addresses the complexities associated with room booking and administration. The aim is to streamline the process of reserving rooms within a hotel, providing a user-friendly platform for customers and staff. This system will have a significant impact, benefiting individuals seeking hassle-free room bookings, hotel organizations aiming for efficient room management, and society by contributing to a more seamless hospitality experience.

Types of Users:

The system caters to two primary user categories: customers and staff members. Customers use the platform to browse available rooms, make reservations, and access their booking history. Staff members, including administrators, use the system to manage room details, view customer bookings, and provide additional services.

Data Collection:

The database collects various data points, including customer details, room availability, booking history, and staff information. Customer data is obtained during the account creation process, while room availability is updated in real-time based on bookings. Staff information is securely stored, ensuring efficient administration.

User Interface:

The user interface is designed to be intuitive and visually appealing. Customers interact with a dashboard that displays available rooms, booking options, and past reservations. Staff members access a dedicated administrative dashboard for managing rooms, viewing bookings, and providing services.

Data Set Size:

While the exact data set size will depend on the scale of hotel operations. The database is designed to efficiently handle a large volume of customer and room-related data. Scalability measures are in place to accommodate the potential expansion of the hotel or the inclusion of additional features in future iterations.

3. Problem Statement:

Managing hotel bookings can be a hassle without a proper system in place, leaving customers uncertain about room availability. To solve this issue, we are developing a Hotel Management System to make it easy for customers to book rooms seamlessly. Our system will focus on a singular hotel, where customers can log in, browse room options, and book their preferred accommodation with ease.

Our user-centric approach aims to provide customers with a straightforward booking experience. Additionally, we aim to empower staff members with the ability to manage room occupancy efficiently. Staff members will have access to room information, and they can check-in guests and process check-outs. Our expanded functionality will provide hotel staff with a comprehensive tool that covers both customer-facing features and internal room management. This way, we ensure a cohesive and effective hotel management system.

4. Objectives:

1. Guest Management: To create a system that maintains accurate and up-to-date guest profiles in a systematic order rather than the traditional book method.
2. Research and analysis: by creating a database we will get a lot of data. With that data, we can make better decisions in fields like resource allocation, decision-making, etc.
3. Improve customer experience: with the help of data, we can give customized suggestions related to hotels to customers.
4. Room Management: The database will aid in Categorizing rooms, track availability, and setting pricing.

5. Software's/Tools Used:

5.1 Frontend:

HTML (HyperText Markup Language):

HTML is the standard markup language used for creating web pages. It defines how a webpage looks like by using tags and attributes. Developers use HTML to design how browsers display different elements such as text, links, images, and media files.

CSS (Cascading Style Sheets):

CSS compliments HTML by specifying the style of your document. Page layouts, colors, fonts, and other stylistic elements are determined using CSS syntax. It emphasizes style and is now in its third version, CSS3. Mastering CSS is crucial for creating visually appealing and well-designed web pages.

JavaScript:

JavaScript is a scripting language that makes web pages dynamic and interactive. It allows developers to create engaging user experiences and execute complex actions. JavaScript makes loading content into a document possible without requiring a full page reload. It is widely used for validation and supporting external applications such as PDF.

EJS (Embedded JavaScript):

EJS is a simple templating language that lets you generate HTML markup with plain JavaScript. It allows you to embed JavaScript code directly within your HTML, making it easier to generate dynamic content. EJS templates are processed by the server to produce HTML that is sent to the client's browser. It's commonly used in Node.js applications with Express.js for rendering dynamic web pages.

Chart.js:

Chart.js is a JavaScript library for creating simple, responsive charts and graphs on web pages. It provides a wide variety of chart types, including line, bar, radar, pie, and more. Chart.js is easy to use and highly customizable, allowing you to create interactive and visually appealing charts with just a few lines of code. It's commonly used in web applications to visualize data and trends, making it easier for users to understand complex data sets.

5.2 Backend:**Node.js:**

Node.js is a popular choice among developers for creating server-side web applications. It is particularly suitable for data-intensive applications, thanks to its asynchronous, event-driven model. Due to this, Node.js makes for an excellent server-side proxy, capable of handling many simultaneous connections in a non-blocking manner. It is especially useful for proxying various services with varying response times or collecting data from multiple source points.

AWS RDS (MySQL):

Amazon Relational Database Service (RDS) is a cloud-based relational database service, which is a product of AWS that works on the basic principles of distributed computing. It scales up, down,

or out, it is cheaper, and fully managed. Thanks to that, you can implement it in the cloud easily at the push of a button.

The availability of several database engines such as MySQL, PostgreSQL, MariaDB, Oracle, SQL Server, and Amazon Aurora on the RDS makes it easy for you to select which one you want to run. Automation is at the heart of RDS, and it performs mission-critical tasks like provisioning, patching, backup, recovery, and scaling, so you can focus more on the development of applications rather than database management.

6. Database Schema:

Table 6.1: Database Schema

Tables	Fields	Relations
1. Customer	id, Name, phoneNumber, address, email	One-to-many with Reviews, Services, Booking, and Receipt
2. Booking	bookingId, checkin, checkout, No_of_people, days, breakfast, payment, customer, roomId, staffId	Many-to-one with Customer, Staff, and room One-to-one with Receipt
3. Login	Email, password, userType	One-to-one with Customer and Staff
4. Receipt	receiptId, checkedin, checkedout, Amount, bookingID	One-to-one with Booking
5. Reviews	reviewID, review, customerID, created_at, rating	Many-to-one with Customer
6. Room	Roomid, Roomtype, price, availability	One-to-many with Booking and Services Many-to-many with Amenities
7. Services	ServiceID, CustomerID, RoomID, Service Type, created_at	Many-to-many with Customer and Room
8. Staff	id, name, phoneNumber, email	One-to-many with Booking
9. Amenities	AmenityID, Amenity_name	Many-to-many with Room
10. RoomAmenities	Room_id, amenity_id	Many-to-one with Room and Amenities

Table 6.2: Views

Views	Fields	Joins
1. booking_people_counts	One_or_More_than_one_person, More_than_three_people, More_than_five_people	None
2. booking_stay_duration_counts	One-or_more_than_one_day,	None

	More_than_two_days, More_than_five_days, more_than_ten_days	
3. booking_summary	Customers_who_had_breakfast, Customers_who_did_noot_have_ breakfast	None
4. booking_year_counts	Year, BookingCount	None
5. booking_year_month_counts	Year, Month, BookingCount	None
6. payment_counts	payment_type, count	UNION selection on booking
7. rating_counts	rating, count	None
8. review_details	ReviewID, review, created_at, rating, Name, email	reviews and customer
9. room_group_counts	RoomGroup, count	None
10. service_details	ServiceID, ServiceType, created_at, RoomID, email, customer_name	services and customer

6.1 Description for each entity: All the tables/entities are in 3NF.

1.Customer:

Description of its attributes:

Customer (

id (PK),

Name,

phoneNumber,

address,

email (UNIQUE)

)

Functional Dependencies:

- id -> Name, phoneNumber, address, email
- email -> id, Name, phoneNumber, address

Columns in table			
Column	Type	Nullable	Indexes
id	int	NO	PRIMARY
Name	varchar(255)	YES	
phoneNumber	varchar(20)	YES	
address	varchar(255)	YES	
email	varchar(45)	YES	email_UNIQUE

Table 6.3: Sample Data for Customer Entity

id	Name	phoneNumber	address	email
12	Bob	3016598768	301 W SIDE DRIVE, Gaithersburg	Bob1234@gmail.com
13	Matt	6709897890	14203 Cedarwood Dr, Cresaptown	matt@gmail.com
14	joy	7896541239	Frostburg	example@gmail.com

2. Booking Entity

Description of attributes:

Booking (
 bookingId (PK),
 checkin,
 checkout,
 No_of_people,
 days,
 breakfast,
 payment,
 customer (FK),
 roomId (FK),
 staffID (FK)
)

Functional Dependencies:

- bookingId -> checkin, checkout, No_of_people, days, breakfast, payment, customer, roomId, staffID
- customer -> bookingId, checkin, checkout, No_of_people, days, breakfast, payment, roomId, staffID
- roomId -> bookingId, checkin, checkout, No_of_people, days, breakfast, payment, customer, staffID
- staffID -> bookingId, checkin, checkout, No_of_people, days, breakfast, payment, customer, roomId

Columns in table			
Column	Type	Nullable	Indexes
bookingId	int	NO	PRIMARY
checkin	date	YES	
checkout	date	YES	
No_of_people	int	YES	
days	int	YES	
breakfast	varchar(5)	YES	
payment	varchar(20)	YES	
customer	int	YES	customer
roomId	int	YES	roomId
staffID	int	YES	staffID

Table 6.4: Sample Data for Booking Entity

bookin gId	checkin	checkout	No_of_pe ople		days	breakfast	payment	custo mer	room Id	staff D
11	2023-11-15	2023-11-16	1		1	on	Credit Card	13	101	1
12	2023-11-16	2023-11-17	2		1	on	Credit Card	12	102	2
14	2023-11-17	2023-11-21	4		4	on	Debit Card	5	101	1
15	2023-11-22	2023-11-25	2		3	on	Credit Card	11	103	4
16	2023-11-22	2023-11-29	2		7	on	Credit Card	1	102	6
18	2023-11-18	2023-11-19	2		1	on	Credit Card	2	102	4
21	2023-12-04	2023-12-06	2		2		Credit Card	8	103	

3. Receipt Entity

Description of attributes:

Receipt (
 receiptId (PK),
 checkedin,
 checkedout,
 Amount,
 bookingID (FK)
)

Functional Dependencies:

- receiptId -> checkedin, checkedout, Amount, bookingID
- bookingID -> receiptId, checkedin, checkedout, Amount

Columns in table			
Column	Type	Nullable	Indexes
receiptId	int	NO	PRIMARY
checkedin	datetime	YES	
checkedout	datetime	YES	
Amount	decimal(10,2)	YES	
bookingID	int	YES	bookingID

Table 6.5: Sample Data for Receipt Entity

receiptId	checkedin	checkedout	Amount	bookingID
2	2023-11-14 18:29:47	2023-11-14 18:29:55	80	8
7	2023-11-22 00:58:06	2023-11-25 00:58:20	120	11

4.Room Entity:

Description of attributes:

Room (

Roomid (PK),

Roomtype,

price,

availability

)

Functional Dependencies:

- Roomid -> Roomtype, price, availability

Columns in table			
Column	Type	Nullable	Indexes
Roomid	int	NO	PRIMARY
Roomtype	varchar(50)	YES	
price	decimal(10,2)	YES	
availability	tinyint(1)	YES	

Table 6.6: Sample Data for Room Entity

Roomid	Roomtype	price	availability
101	Queen	50	1
102	King	80	1
103	Double	40	1

5.Services Entity:

Description of attributes:

Services (

ServiceID (PK),

CustomerID (FK),

RoomID (FK),

ServiceType,

created_at

)

Functional Dependencies:

- ServiceID -> CustomerID, RoomID, ServiceType, created_at
- CustomerID -> ServiceID, RoomID, ServiceType, created_at
- RoomID -> ServiceID, CustomerID, ServiceType, created_at

Columns in table			
Column	Type	Nullable	Indexes
ServiceID	int	NO	PRIMARY
CustomerID	int	YES	CustomerID
RoomID	int	YES	RoomID
ServiceType	varchar(50)	YES	
created_at	timestamp	YES	

Table 6.7: Sample Data for Service Entity

ServiceID	CustomerID	RoomID	ServiceType	created_at
1	12	101	laundry	2023-11-14 20:48:43

6.Staff Entity:

Description of attributes:

Staff (

id (PK),

name,

phoneNumber,

email (UNIQUE)

)

Functional Dependencies:

- id -> name, phoneNumber, email
- email -> id, name, phoneNumber

Columns in table			
Column	Type	Nullable	Indexes
id	int	NO	PRIMARY
name	varchar(255)	YES	
phoneNumber	varchar(20)	YES	
email	varchar(45)	YES	email_UNIQUE

Table 6.7: Sample Data for Staff Entity

id	name	phoneNumber	email
4	Sam	3016591275	sam@gmail.com
5	bob	3547561238	Bh@gmail.com

7.Review Entity:

Description of attributes:

Reviews (

reviewID (PK),

review,

customerID (FK),

created_at,

rating

)

Functional Dependencies:

- reviewID -> review, customerID, created_at, rating
- customerID -> reviewID, review, created_at, rating

Columns in table			
Column	Type	Nullable	Indexes
reviewID	int	NO	PRIMARY
review	text	YES	
customerID	int	YES	customerID
created_at	timestamp	YES	
rating	int	YES	

Table 6.8: Sample Data for Review Entity

reviewID	review	customerID	created_at	rating
	the rooms are good, had good time and stay			
2	at frostburg inn	12	2023-11-14 22:34:55	4
3	Good	14	2023-11-17 00:26:20	1

8.Amenities Entity:

Description of attributes:

Amenities (

amenity_id (PK),

amenity_name

)

Functional Dependencies:

- amenity_id → amenity_name

Columns in table			
Column	Type	Nullable	Indexes
amenity_id	int	NO	PRIMARY
amenity_name	varchar(50)	NO	

Table 6.9: Sample Data for Amenities Entity

amenity_id	amenity name
1	TV
2	Air conditioning
3	Telephone
4	microwave

9.RoomAmenities Entity:

Description of attributes:

RoomAmenities (

room_id (FK),

amenity_id (FK),

PRIMARY KEY (room_id, amenity_id)

)

Functional Dependencies:

- room_id, amenity_id -> (no additional dependencies as it is a junction table)

Columns in table			
Column	Type	Nullable	Indexes
room_id	int	NO	PRIMARY
amenity_id	int	NO	PRIMARY, amenity_id

Table 6.10: Sample Data for RoomAmenities Entity

room_id	amenity_id
101	1
102	1
103	2
103	3

10. Login Entity:

Description of attributes:

Login (

Email,

Password,

UserType,

PRIMARY KEY (Email)

)

Columns in table			
Column	Type	Nullable	Indexes
◇ Email	varchar(255)	NO	PRIMARY, Email_UNIQUE
◇ password	varchar(20)	YES	
◇ userType	varchar(50)	YES	

Table 6.11: Sample Data for Login Entity

Email	password	userType
Bh@gmail.com	1234	Staff
example@gmail.com	9876	Customer
lvinay@gmail.com	dddd	Staff
matt@gmail.com	dddd	Customer
navyasree809@gmail.com	dddd	Customer

6.2 Description for each views:

1. booking_people_counts:

Table 6.12: booking_people_counts - View

One_or_More_than_one_person	More_than_three_people	More_than_five_people
142	98	8

2. booking_stay_duration_counts:

Table 6.13: booking_stay_duration_counts - View

One_or_More_than_one_day	more_than_two_days	more_than_five_days	more_than_ten_days
156	71	21	0

3. booking_summary:

Table 6.14: booking_summary - View

Customers_who_had_breakfast	Customers_who_did_not_have_breakfast
220	28

4. booking_year_counts:

Table 6.15: booking_year_counts - View

Year	BookingCount
2023	178
2024	70

5. booking_year_month_counts:

Table 6.16: booking_year_month_counts - View

Year	Month	BookingCount
2023	4	18
2023	5	17
2023	6	22
2023	7	20
2023	8	22
2023	9	17
2023	10	22
2023	11	21
2023	12	19
2024	1	19
2024	2	17
2024	3	15
2024	4	10
2024	5	4
2024	6	1
2024	7	1
2024	9	3

6. payment_counts:

Table 6.17: payment_counts - View

payment_type	count
Cash	154
Credit Card	51
Debit Card	42

7. rating_counts:

Table 6.18: rating_counts - View

rating	count
5	46
1	36
4	47
2	42
3	30

8. review_details:

Table 6.19: review_details - View

reviewID	review	created_at	rating	Name	email
1	Excellent	26-10-2023 00:00	5	Iggie Baudinelli	ibaudinellij1@simplemachines.org
2	Excellent	13-03-2024 00:00	5	Samuele Gowdy	sgowdypi@google.de
3	Excellent	06-02-2024 00:00	5	Mahala Thridgould	mthridgouldi0@cisco.com
4	Bad	20-12-2023 00:00	1	Bartlet Furmedge	bfurmedgere@odnoklassniki.ru
5	Excellent	15-11-2023 00:00	5	Cristian Dannohl	cdannohlmi@furl.net
6	Good	06-12-2023 00:00	4	Durante Douch	ddouch23@soundcloud.com
7	Bad	30-09-2023 00:00	2	Corinne Sueter	csueter0@ning.com
8	Excellent	21-12-2023 00:00	5	Alister Abeau	aabeauqa@bloglines.com
9	Excellent	07-12-2023 00:00	5	Willie Shearman	wshearmanmt@yahoo.co.jp
10	Bad	08-11-2023 00:00	1	Jeannie Ackrill	jackrillc@addtoany.com
11	Excellent	02-12-2023 00:00	5	Corilla Merryweather	cmerryweatherrf@statcounter.com

9. room_group_counts:

Table 6.20: room_group_counts - View

roomGroup	count
-----------	-------

Group 1	73
Group 2	80
Group 3	95

10. service_details:

Table 6.21: service_details - View

ServiceID	serviceType	created_at	RoomID	email	customer_name
112	Iron	26-01-2024 00:00	102	rruddiman0@princeton.edu	Reeva Ruddiman
224	Iron	17-05-2024 00:00	103	treeson1@trellian.com	Tatum Reeson
1003	laundry	08-04-2024 18:50	107	treeson1@trellian.com	Tatum Reeson
1004	laundry	08-04-2024 19:15	107	treeson1@trellian.com	Tatum Reeson
1005	laundry	08-04-2024 19:17	107	treeson1@trellian.com	Tatum Reeson
335	Room cleaning	05-09-2024 00:00	104	yshann2@constantcontact.co m	Yolanda Shann
446	Room cleaning	25-12-2024 00:00	105	sbramall3@ibm.com	Samara Bramall
1006	laundry	08-04-2024 19:25	101	sbramall3@ibm.com	Samara Bramall
557	Ice Cubes	15-04-2025 00:00	106	mlessmare4@moonfruit.com	Melissa Lessmare

6.2 ER Diagram:

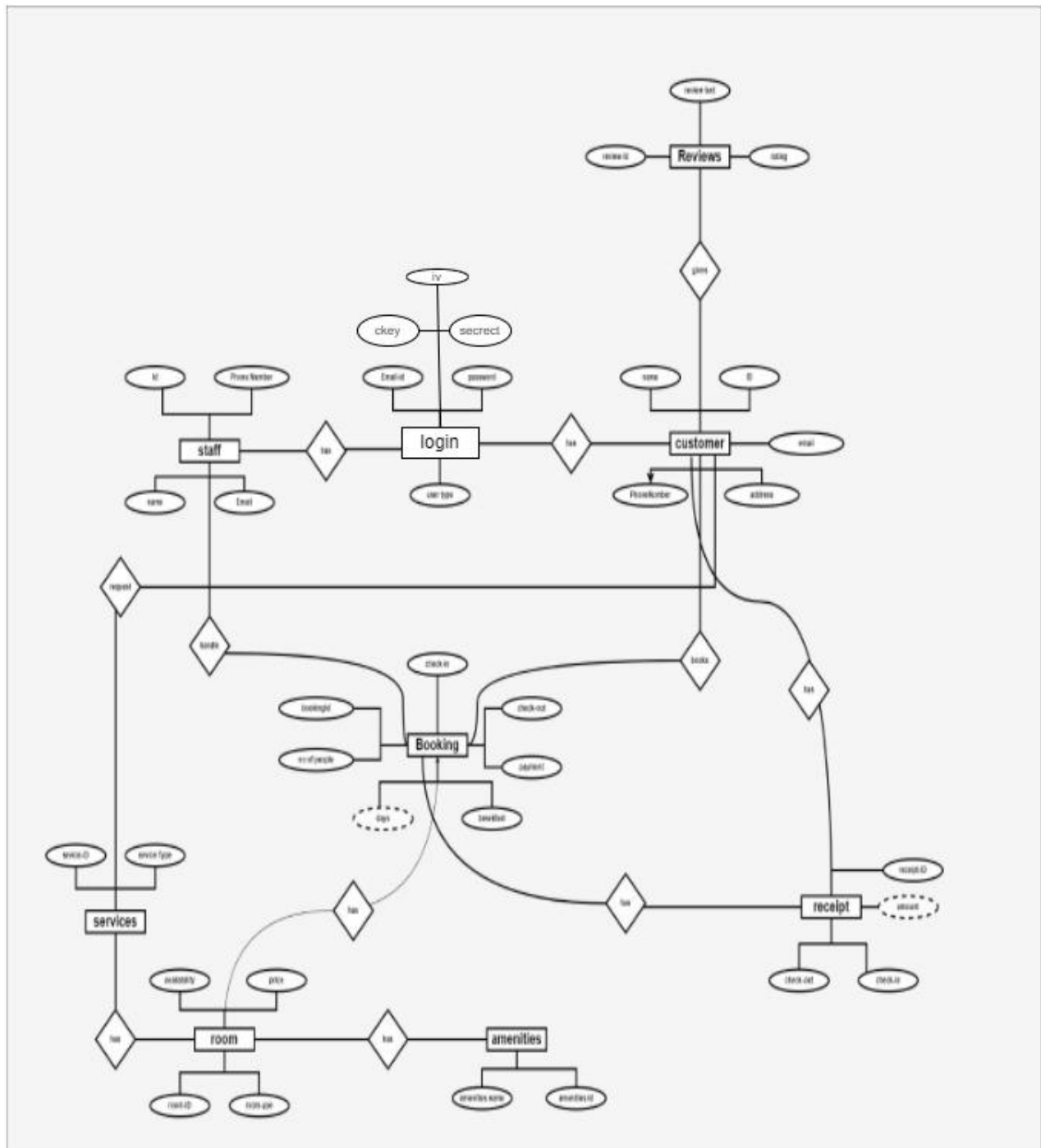


Fig 6.1: ER Diagram

6.3 Relational Diagram:

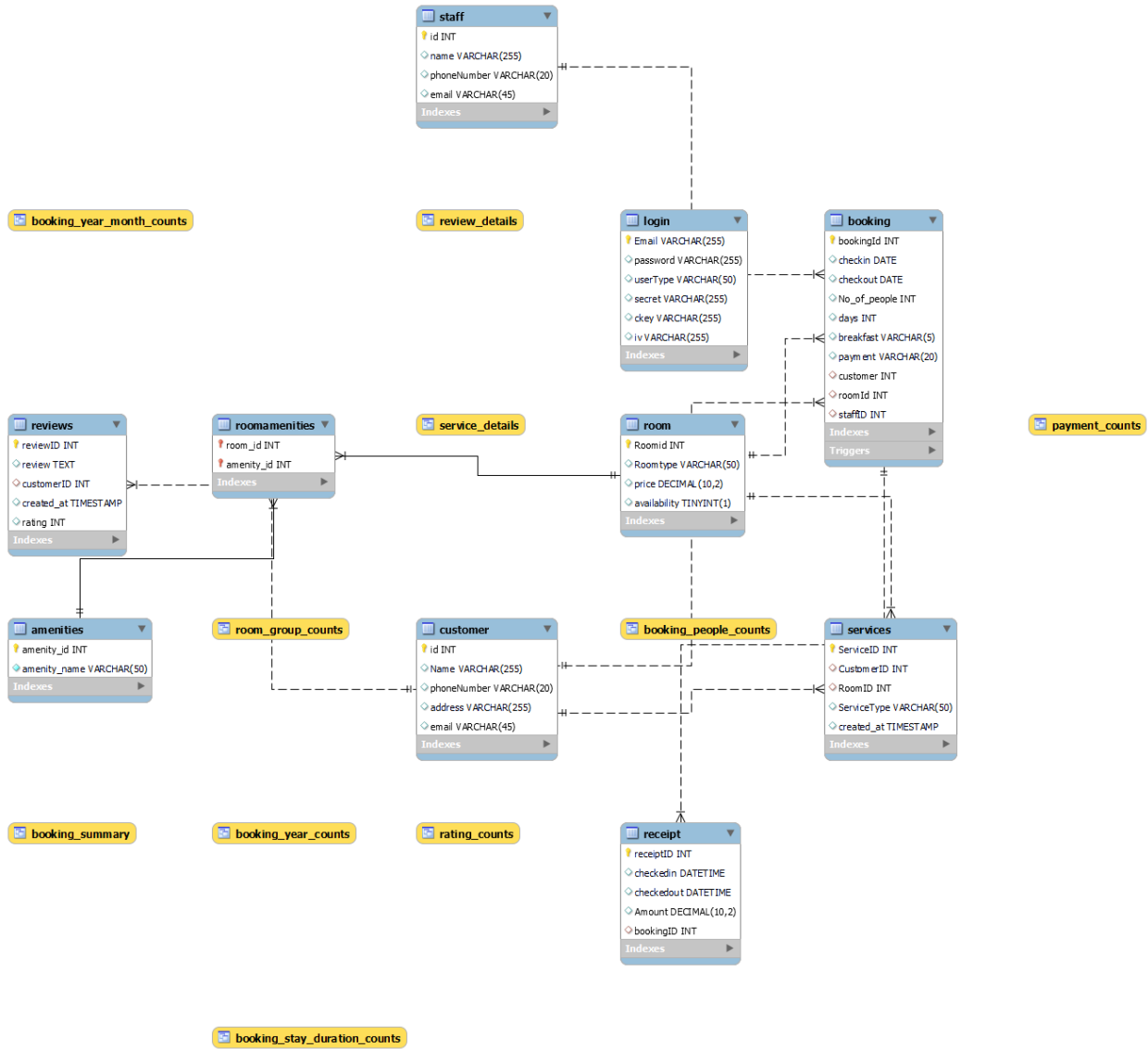


Fig 6.2: Relational Diagram

7. User Interface Design:

We have built our user interface using HTML, CSS, and JavaScript.

7.1 Use case diagram:

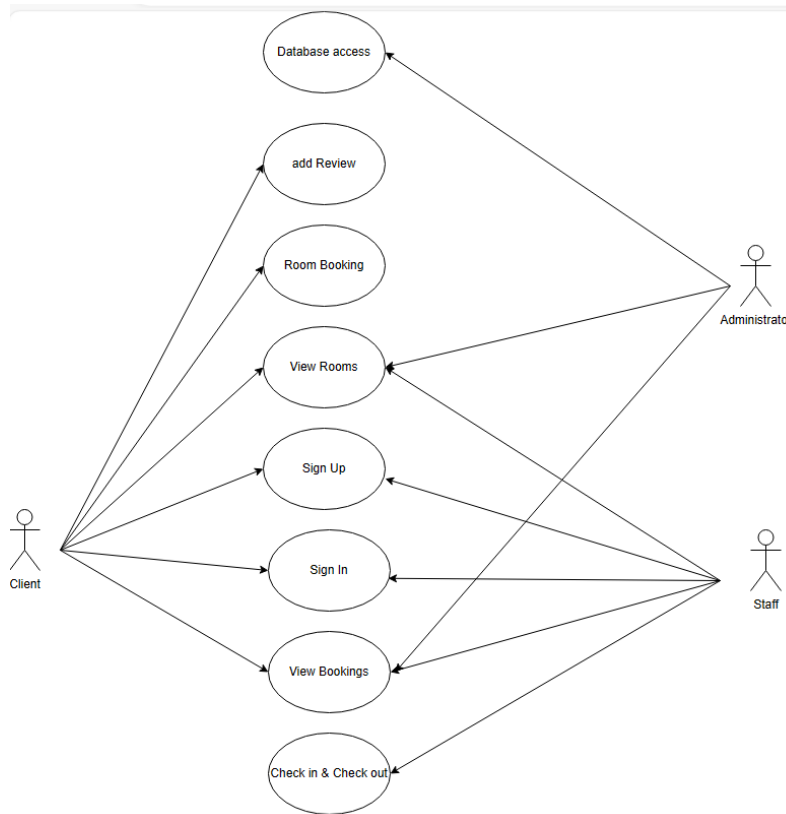


Fig 7.1: Use Case Diagram

7.1.1 Signup and Login

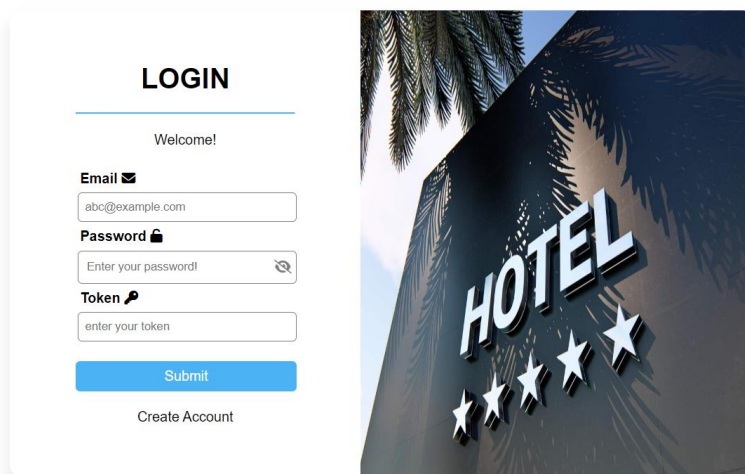
User Stories: Client

Client Needs: The client requires a robust and user-friendly interface for creating and accessing accounts, enabling a seamless sign-up and sign-in process on the primary booking webpage.

Functionalities:

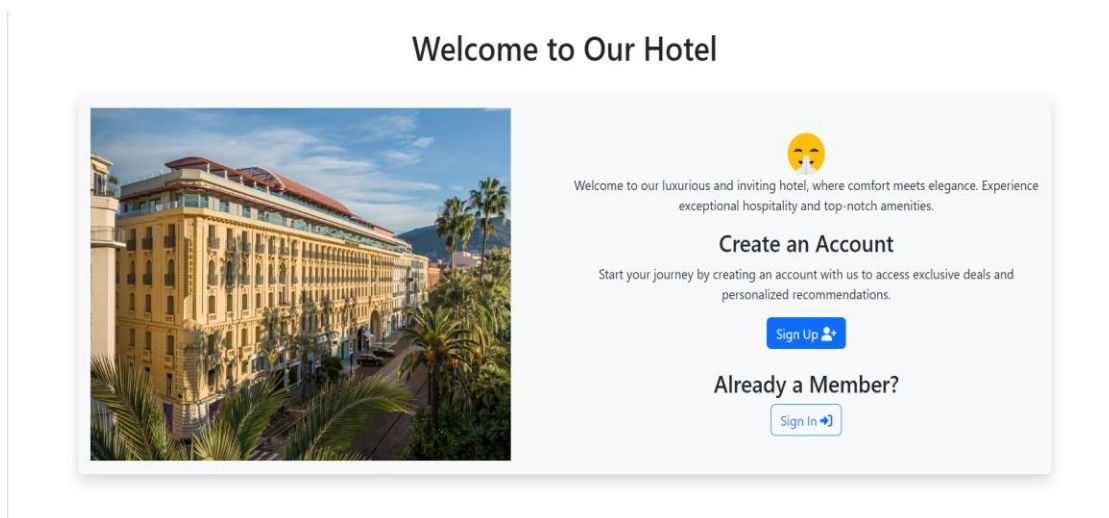
1. **Account Creation:** When a user lands on the "Create Account" page, they will find a user-friendly form to enter their necessary details. The page features a prominent "Sign Up" or "Create Account" button, which initiates the submission of registration details when clicked.

2. **Successful Registration:** Upon successful registration, the user is automatically redirected to the login page. The registration process includes clear error handling and providing instructive messages for issues like duplicate email addresses or other errors.
3. **Login Process:** The login page offers fields for users to enter their registered email address and password.
4. **User Experience:** The entire registration process is designed to be user-friendly, intuitive, and focused on delivering a positive experience for the user. The created account seamlessly unlocks access to the room booking page.



The image shows a login dashboard with a white background and a blue header. The header contains the word "LOGIN" in bold. Below the header, there is a "Welcome!" message. The main form has three input fields: "Email" with a placeholder "abc@example.com", "Password" with a placeholder "Enter your password!" and a toggle icon, and "Token" with a placeholder "enter your token". A blue "Submit" button is at the bottom of the form. Below the button is a link "Create Account". To the right of the form is a large image of a hotel building with a sign that says "HOTEL" and five stars.

Fig 7.2: Login Dashboard



The image shows a sign up dashboard with a white background and a blue header. The header contains the text "Welcome to Our Hotel". Below the header, there is a large image of a hotel building. To the right of the image, there is a "Welcome" message with a yellow smiley face icon. Below the message, there is a "Create an Account" section with a "Sign Up" button. Below the "Sign Up" button, there is an "Already a Member?" section with a "Sign In" button.

Fig 7.3: Sign Up Dashboard

Fig 7.3: Sign Up Form

7.1.2 Customer Dashboard

User Stories: Client/Customer

Client Needs: The customer needs an easy-to-use and feature-packed dashboard that allows him to book rooms, view past bookings, select breakfast options, and provide room feedback.

Functionalities:

1. **Dashboard Landing:** After logging in successfully, the User will be directed to a personalized dashboard that displays all the necessary information relevant to him.
2. **Room Booking:** The dashboard has a user-friendly interface that makes it easy for the user to browse available rooms based on categories and book them. Users will be able to select the check-in and check-out dates and room preferences.
3. **View Previous Bookings:** User will be able to access a section on the dashboard that displays details of my past bookings, including the check-in and check-out dates.
4. **Select Breakfast:** During the booking process, there is an option for the users to select their breakfast preferences.
5. **Review Room:** Users should be able to provide feedback by submitting reviews of rooms they stayed in. This will help to improve the overall user experience.
6. **Logout:** There should be an option for users to log out securely to ensure the privacy and security of their account.



Fig 7.4: Navigation Section of Customer Dashboard


Home

About Us

Rooms


VAN

Our Rooms



Deluxe Room 1

A deluxe hotel room offers a step above the standard accommodation options, making it an upgraded experience. If you book a deluxe room, you can expect larger and more tastefully designed spaces, often nicer furnishings,



Deluxe Room 1

A deluxe hotel room offers a step above the standard accommodation options, making it an upgraded experience. If you book a deluxe room, you can expect larger and more tastefully designed spaces, often nicer furnishings, premium bedding, a more elaborate seating area or extra features like a tub or fireplace

Price \$: 50 /night

Room Number #: 101


Book Now

Amenities:

• TV


• AC

• Fan



Deluxe Room 2

A deluxe hotel room offers a step above the standard accommodation options, making it an upgraded experience. If you book a deluxe room, you can expect larger and more tastefully designed spaces, often nicer furnishings,



Deluxe Room 2

A deluxe hotel room offers a step above the standard accommodation options, making it an upgraded experience. If you book a deluxe room, you can expect larger and more tastefully designed spaces, often nicer furnishings, premium bedding, a more elaborate seating area or extra features like a tub or fireplace

Price \$: 50 /night

Room Number #: 102

Book Now

Amenities:


• TV

• AC

• Fan


• Microwave

• Hair dryer



Deluxe Room 3

A deluxe hotel room offers a step above the standard accommodation options, making it an upgraded experience. If you book a deluxe room, you can expect larger and more tastefully designed spaces, often nicer furnishings,



Deluxe Room 3

A deluxe hotel room offers a step above the standard accommodation options, making it an upgraded experience. If you book a deluxe room, you can expect larger and more tastefully designed spaces, often nicer furnishings, premium bedding, a more elaborate seating area or extra features like a tub or fireplace

Price \$: 50 /night

Room Number #: 103

Book Now

Amenities:

• TV

• AC

• Fan

• Hair dryer

Fig 7.5: Display of Available Rooms

29

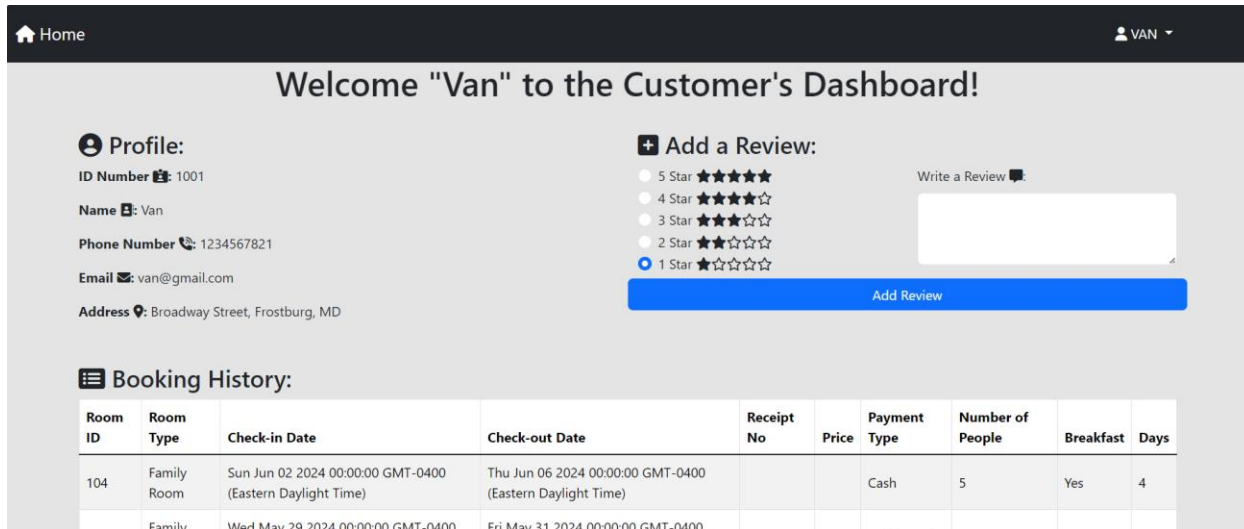


Fig 7.6: Webpage to show Booking History

7.1.3 Admin page

User Stories: Staff/admin

Staff Needs: The staff/admin requires a robust dashboard and an admin page to efficiently manage room bookings, check-in and check-out procedures, and maintain room information.

Functionalities:

1. **Dashboard Overview:** Upon login, the staff is directed to a comprehensive dashboard providing an overview of current room statuses, upcoming bookings, and other relevant information.
2. **View Room Bookings:** The admin page includes a section for viewing all current and upcoming room bookings, with details such as customer names, check-in/out dates, and room types.
3. **Check-in/Check-out:** Staff members/admin can check in on customers arriving at the hotel and check out customers leaving. This involves updating room statuses and managing key logistics.
4. **Access Customer Reviews:** The admin page includes a section dedicated to customer reviews. Here, the staff can view the feedback, ratings, and comments provided by guests. By analyzing this information, the staff can better understand customer satisfaction and address any concerns.
5. **View Available Services:** The admin page also includes a section for displaying all the available services offered by the hotel and admin can also add the services to the available service list.
6. **View Available Amenities:** The admin page also includes a section for displaying all the amenities for a room and admin can also add amenities to existing amenities list.
7. **Service Management:** The admin page includes a section that enables staff to add new services.

8. **Amenities Management:** This is a section which displays all amenities available for each room, with options for staff to add new amenities or update existing ones based on guest feedback and hotel upgrades.
9. **Logout:** The staff can securely log out from the dashboard, ensuring data security.

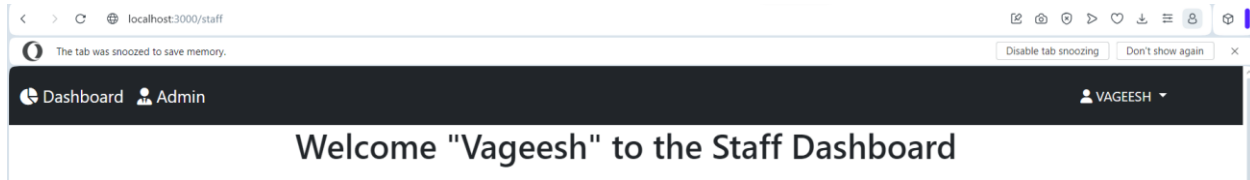


Fig 7.7: Navigation Section of Staff Dashboard

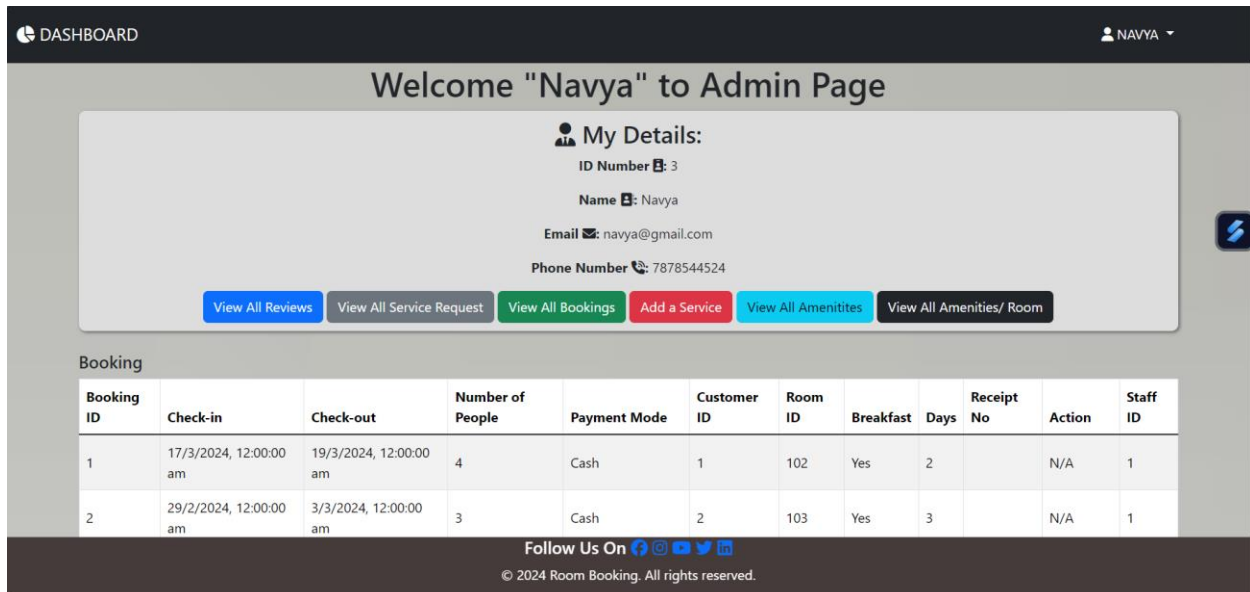


Fig 7.8: Webpage to View all Bookings, checkin and checkout Date

DASHBOARD NAVYA

Welcome "Navya" to Admin Page

My Details:

ID Number: 3

Name: Navya

Email: navya@gmail.com

Phone Number: 7878544524

[View All Reviews](#)
[View All Service Request](#)
[View All Bookings](#)
[Add a Service](#)
[View All Amenities](#)
[View All Amenities/ Room](#)

Services

Service ID	Service Type	Created At	Room ID	Customer Name	Customer Email
1	Iron	Sat Oct 07 2023 00:00:00 GMT-0400 (Eastern Daylight Time)	101	Rosalia Sweetlove	rsweetlove8@toplist.cz
2	Iron	Sun Oct 08 2023 00:00:00 GMT-0400 (Eastern Daylight Time)	101	Ki Hammerson	khammersonh@myspace.com
3	Iron	Mon Oct 09 2023 00:00:00 GMT-0400 (Eastern Daylight Time)	101	Sayers Stiven	sstivenq@exblog.jp

Follow Us On [f](#) [i](#) [t](#) [y](#) [l](#)

© 2024 Room Booking. All rights reserved.

Fig 7.9: Webpage to View all Services

DASHBOARD NAVYA

Welcome "Navya" to Admin Page

My Details:

ID Number: 3

Name: Navya

Email: navya@gmail.com

Phone Number: 7878544524

[View All Reviews](#)
[View All Service Request](#)
[View All Bookings](#)
[Add a Service](#)
[View All Amenities](#)
[View All Amenities/ Room](#)

Reviews

Review ID	Customer Name	Email	Rating	Date	Review
1	Iggie Baudinelli	ibaudinelli1@simplemachines.org	5	Thu Oct 26 2023 00:00:00 GMT-0400 (Eastern Daylight Time)	Excellent
2	Samuele Gowdy	sgowdypi@google.de	5	Wed Mar 13 2024 00:00:00 GMT-0400 (Eastern Daylight Time)	Excellent
3	Mahala Thridgould	mthridgouldi@cisco.com	5	Tue Feb 06 2024 00:00:00 GMT-0500 (Eastern Standard Time)	Excellent
4	Partlet Sumedee	bfumedee@edncklscnili.ru	1	Wed Dec 20 2023 00:00:00 GMT-0500 (Eastern Standard Time)	Bad

Follow Us On [f](#) [i](#) [t](#) [y](#) [l](#)

© 2024 Room Booking. All rights reserved.

Fig 7.10: Webpage to View all Reviews

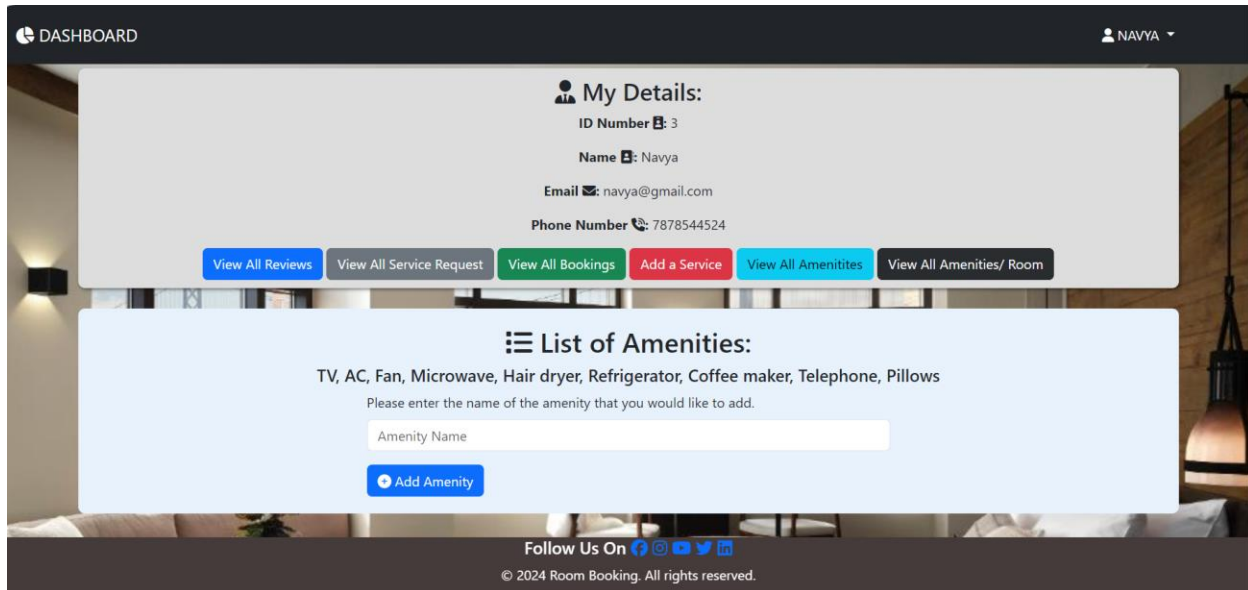


Fig 7.11: Webpage to View all Amenities

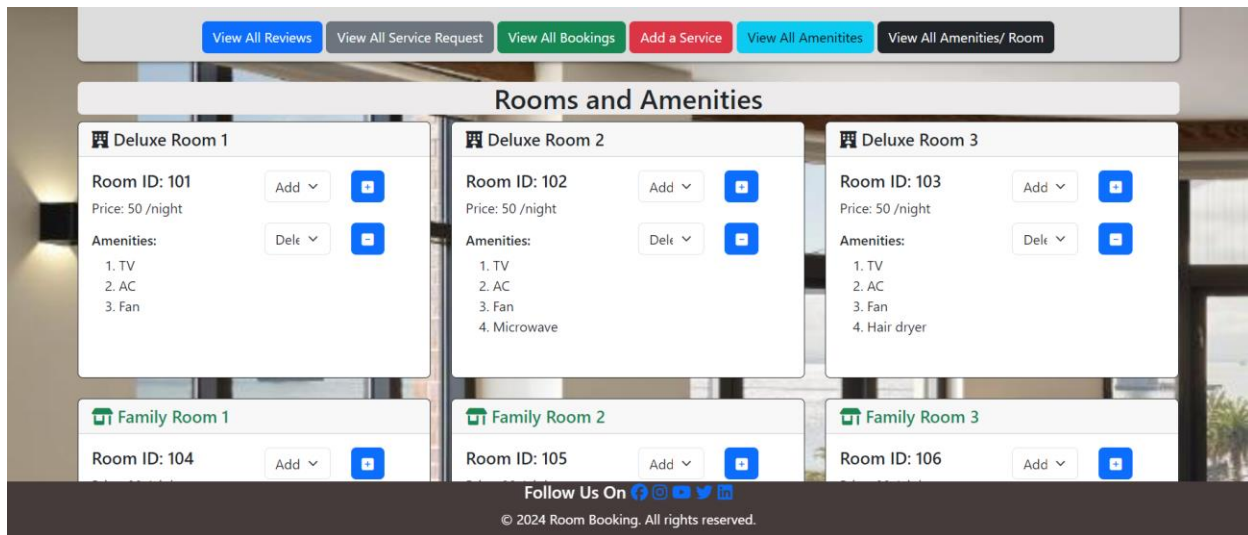


Fig 7.12: Webpage to add Amenities to the List

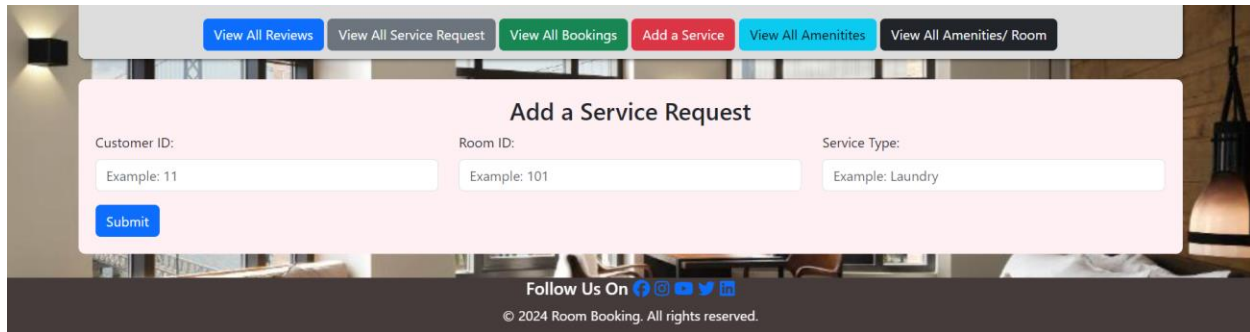
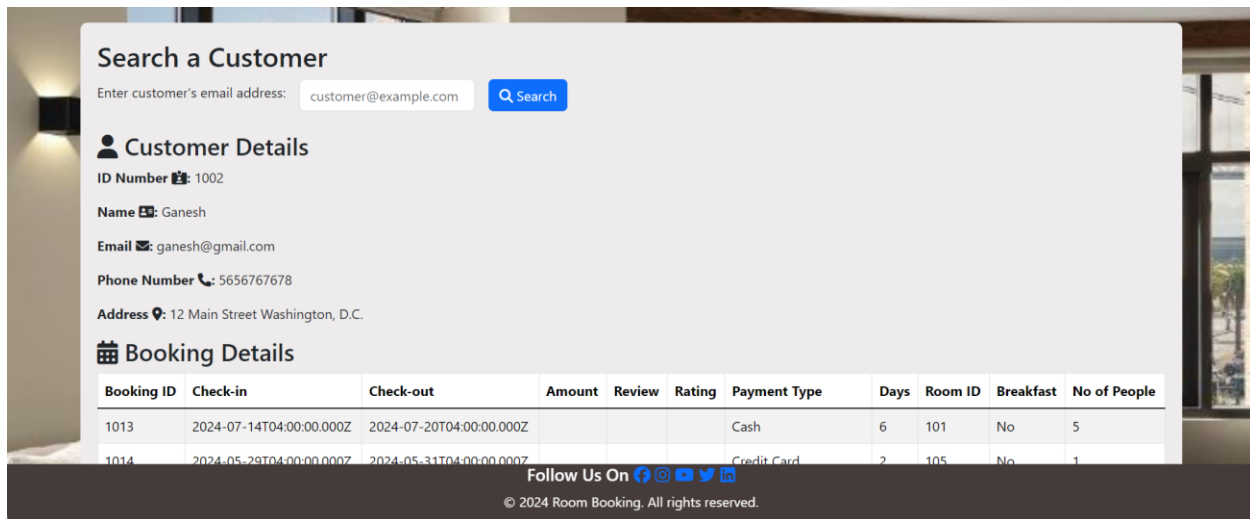


Fig 7.13: Webpage to add Service to the List



Booking ID	Check-in	Check-out	Amount	Review	Rating	Payment Type	Days	Room ID	Breakfast	No of People
1013	2024-07-14T04:00:00.000Z	2024-07-20T04:00:00.000Z				Cash	6	101	No	5
1014	2024-05-29T04:00:00.000Z	2024-05-31T04:00:00.000Z				Credit Card	2	105	No	1

Fig 7.14: Webpage to Search a Customer by Email

7.1.4 Staff/Admin Dashboard

User Stories: Staff/Admin

Staff Needs: To ensure operational efficiency and customer satisfaction, the staff/admin requires a well-organized dashboard that facilitates easy management of room bookings and provides data for business analysis.

Functionalities:

1. **Dashboard Overview:** Upon login, the staff is directed to a comprehensive dashboard providing a real-time overview of hotel operations. Displays current room occupancy status, upcoming bookings, revenue trends, and other key performance indicators for business analysis.
2. **Manage Room Bookings:** staff can visually see booking trends based on month.
 - **Purpose:** Allows staff to view, edit, and manage room bookings.
 - **Features:** List of all current and upcoming bookings with options to modify booking details, customer names, check-in/out dates, and room preferences.

3. **Business Analysis:** Staff can see visual representations of data such as revenue trends, occupancy rates, booking patterns, and guest demographics to help in business planning and strategy. Which provides data insights for decision-making.
4. **Customer Reviews and Feedback:** The admin page includes a chart for view rating trends which helps staff monitor and improve service quality.
 - **Features:** A section dedicated to viewing guest feedback and reviews, sortable by date, room, or rating.
5. **Logout:** Secure logout option for staff to safely exit their dashboard session, safeguarding sensitive information and access controls.

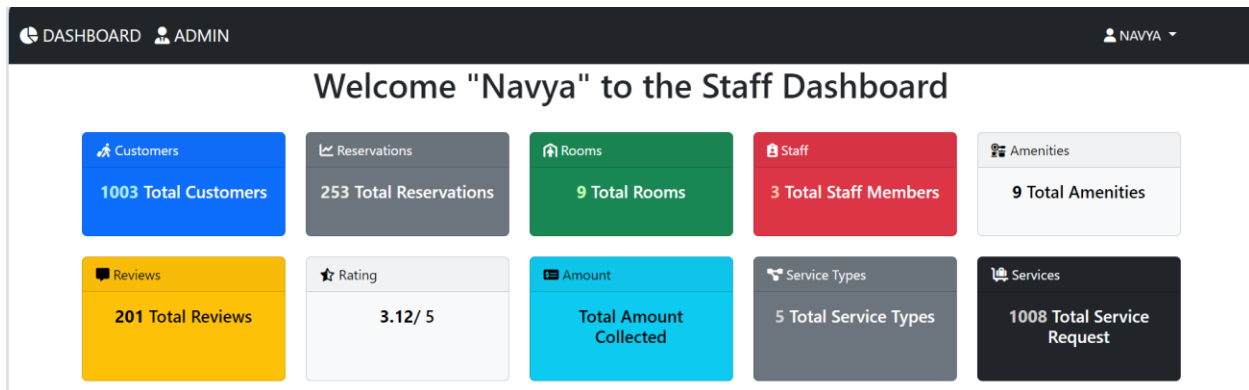


Fig 7.15: Overview of staff Dashboard

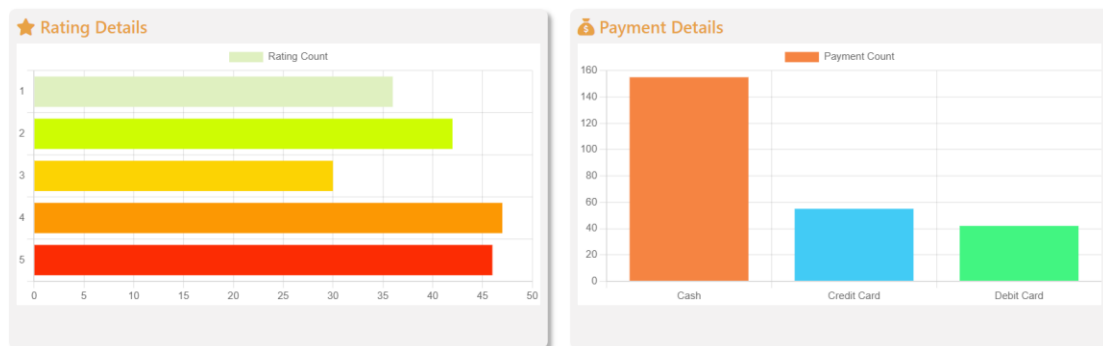


Fig 7.16: Graph to Show Rating and Payment Details

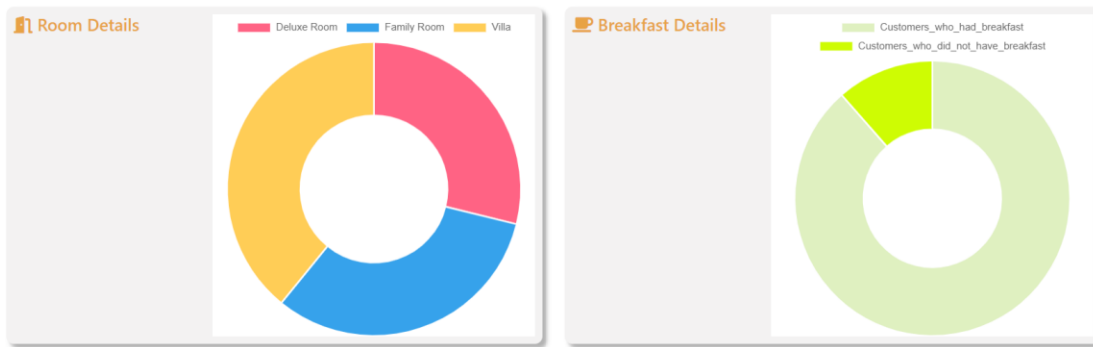


Fig 7.17: Graph to Show Room and Breakfast Details

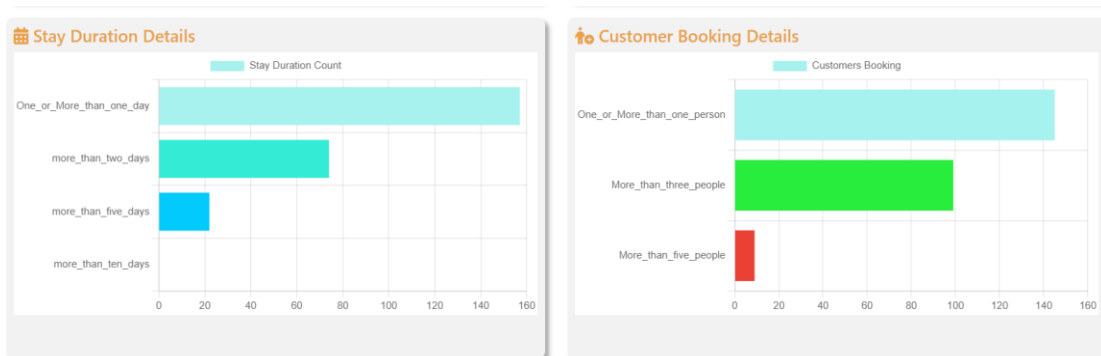


Fig 7.18: Graph to Show Saty Duration and Booking Details

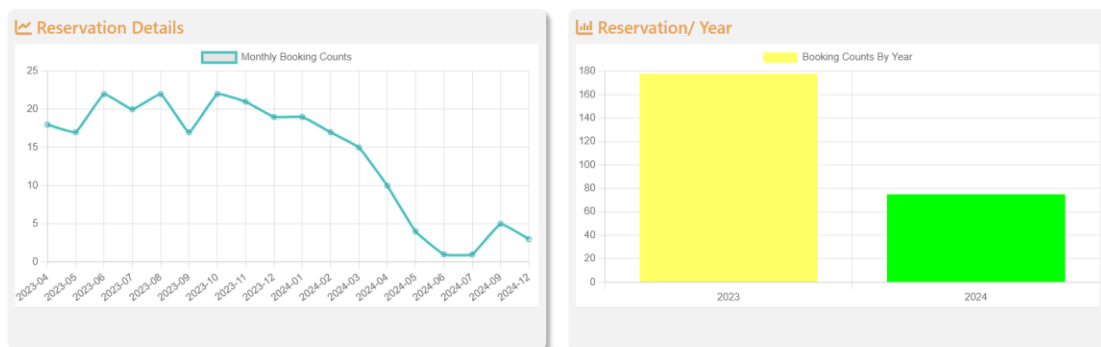


Fig 7.19: Graph to Show Reservation Trends

7.2 Queries by Functionalities:

Query 1: Insert into Customer Table

'INSERT INTO customer (name, phoneNumber, address,email) VALUES (?, ?, ?,?)',

[name, phoneNumber, address, email]

Query 2: Insert into Login Table

'INSERT INTO login (email, password ,userType) VALUES (?, ?, ?)',
[email, password, type]

Query 3: Insert into Staff Table

'INSERT INTO staff (name, phoneNumber,email) VALUES (?, ?,?)',
[name, phoneNumber, email]

Query 4: validate user credentials

'SELECT userType FROM login WHERE email = ? AND password = ?',
[email, password],

Query 5: Select available rooms along with amenities available in each room

```
`SELECT
    Room.Roomid,
    Room.Roomtype,
    Room.price,
    GROUP_CONCAT(Amenities.amenity_name ORDER BY Amenities.amenity_name SEPARATOR ' ') AS
amenities
FROM
    Room
JOIN
    RoomAmenities ON Room.Roomid = RoomAmenities.room_id
```

JOIN

Amenities ON RoomAmenities.amenity_id = Amenities.amenity_id

WHERE

Room.availability = 1

GROUP BY

Room.Roomid`

Query 6: Insert into booking table

'INSERT INTO booking (checkin,checkout,No_of_people,breakfast,payment,customer,roomId) VALUES
(?, ?,?,?,?,?),'

[checkInDate, checkOutDate, numberOfPeople, breakfastPreference, paymentMode, customerId,
roomId]

Query 7: Select Customers booking history

'SELECT * FROM booking WHERE customer = ? ORDER BY checkin DESC', [customerId]

Query 8: Insert into Reviews Table

'INSERT INTO reviews(review,customerID,rating) VALUES (?,?),'

[review, customerId, rating]

Query 9: Display all booking along with checkin, check out status and receipt numbers

'SELECT b.bookingId, b.checkin, b.checkout, b.No_of_people, b.days, b.breakfast,
b.payment, b.customer, b.roomId, b.staffID, r.receiptId, r.checkedin, r.checkedout
FROM booking b LEFT JOIN receipt r ON b.bookingId=r.bookingID'

Query 10: Insert into Receipt table after checkin

```
'INSERT INTO receipt(checkedin, bookingID) VALUES (?, ?)'
```

Query 11: call calculateAndUpdatePrice Procedure after checkin

```
'CALL CalculateAndUpdatePrice(?, ?)',  
      [bookingId, days]
```

Query 12: calculateAndUpdatePrice Procedure

```
DELIMITER //
```

```
DROP PROCEDURE IF EXISTS CalculateAndUpdatePrice;
```

```
CREATE PROCEDURE CalculateAndUpdatePrice(
```

```
    IN bookid INT,
```

```
    IN days INT
```

```
)
```

```
BEGIN
```

```
    DECLARE roomPrice DECIMAL(10, 2);
```

```
DECLARE totalPrice DECIMAL(10, 2);
```

```
-- Get the price from the room table based on bookingId
```

```
SELECT r.price INTO roomPrice
```

```
FROM room r
```

```
INNER JOIN booking b ON b.roomId = r.Roomid
```

```
WHERE b.bookingId = bookid;
```

```
-- Debug: Print roomPrice
```

```

-- SELECT roomPrice AS debug_roomPrice;
SELECT bookid AS debug_roomPrice;
-- Calculate the total price

SET totalPrice = roomPrice * days;

-- Debug: Print totalPrice
-- Debug: Print bookingId

-- Update the receipt table with the calculated price for each row
UPDATE receipt
SET Amount = totalPrice
WHERE bookingID = bookid;

END //
DELIMITER ;

```

Query 13: Update booking table with the staff who is responsible for checkin

```

UPDATE booking b SET b.staffID=? WHERE b.bookingid =?
[customerId, bookingId]

```

Query 14: Update receipt table during checkout

```

'UPDATE receipt SET checkedout=? WHERE bookingID=?',
[formattedDate, bookingId]

```

Query 15: Select all the services the user booked


```
SELECT * FROM services
```

Query 16: Select all reviews with customer name

```
'SELECT r.review,r.created_at,r.rating,c.Name FROM reviews r JOIN customer c ON c.id=r.customerID'
```

Query 17: Trigger to calculate number of days for each booking

```
CREATE TRIGGER calculate_days_trigger
BEFORE INSERT ON booking
FOR EACH ROW
SET NEW.days = DATEDIFF(NEW.checkout, NEW.checkin);
```

Query 18: Duplicate Booking check logic

```
'SELECT * FROM booking WHERE roomId = ? AND ((checkin BETWEEN ? AND ?) OR (checkout BETWEEN ?
AND ?))',
[roomId, checkInDate, checkOutDate, checkInDate, checkOutDate]
```

7.3 Query Optimization and Indexes

Query 1

```
SELECT Room.Roomid, Room.Roomtype, Room.price,
GROUP_CONCAT(Amenities.amenity_name ORDER BY Amenities.amenity_name SEPARATOR ', ') AS
amenities FROM Room JOIN
RoomAmenities ON Room.Roomid = RoomAmenities.room_id
JOIN
Amenities ON RoomAmenities.amenity_id = Amenities.amenity_id
WHERE Room.availability = 1
GROUP BY Room.Roomid;
```

Before Creating index:

70 22:21:32 SELECT Room.Roomid, Room.Roomtype, Room.price, GROUP_CONCAT(Amenities.amenity_name ORD... 0 row(s) returned 0.078 sec / 0.000 sec

After Creating index:

72 22:25:48 SELECT Room.Roomid, Room.Roomtype, Room.price, GROUP_CONCAT(Amenities.amenity_name ORD... 0 row(s) returned 0.000 sec / 0.000 sec

Query 2

```
SELECT b.bookingId, b.checkin, b.checkout, b.No_of_people, b.days, b.breakfast,
b.payment, b.customer, b.roomId, b.staffID, r.receiptId, r.checkedin, r.checkedout
FROM booking b LEFT JOIN receipt r ON b.bookingId=r.bookingId;
```

Before Creating index:

73 22:36:28 SELECT b.bookingId, b.checkin, b.checkout, b.No_of_people, b.days, b.breakfast, b.payment, b.customer,... 1000 row(s) returned 0.078 sec / 0.015 sec

After Creating Index:

75 22:44:33 SELECT b.bookingId, b.checkin, b.checkout, b.No_of_people, b.days, b.breakfast, b.payment, b.customer,... 1000 row(s) returned 0.000 sec / 0.016 sec

Query 3

```
SELECT * FROM login;
```

Before Creating Index:

77 22:47:45 SELECT * FROM hotel_management.login LIMIT 0, 1000 1000 row(s) returned 0.032 sec / 0.000 sec

After Creating Index:

79 22:48:49 SELECT * FROM hotel_management.login LIMIT 0, 1000 1000 row(s) returned 0.000 sec / 0.000 sec

Query 4

```
SELECT r.review,r.created_at,r.rating,c.Name FROM reviews r JOIN customer c ON c.id=r.customerID;
```

Before Creating Index:

3 23:05:59 SELECT r.review,r.created_at,r.rating,c.Name FROM reviews r JOIN customer c ON c.id=r.customerID LIM... 201 row(s) returned 0.016 sec / 0.000 sec

After Creating Index:

7 23:08:28 SELECT r.review,r.created_at,r.rating,c.Name FROM reviews r JOIN customer c ON c.id=r.customerID LIM... 201 row(s) returned 0.000 sec / 0.000 sec

Query 5

```
SELECT booking.*, receipt.Amount, customer.name AS customerName,
customer.phoneNumber, customer.address, customer.email, room.Roomtype FROM booking
LEFT JOIN receipt ON booking.bookingId = receipt.bookingId LEFT JOIN customer ON
booking.customer = customer.id LEFT JOIN room ON booking.roomId = room.Roomid WHERE
booking.customer = ? ORDER BY booking.checkin DESC
```

Optimized version

```
SELECT booking.checkin, booking.checkout, booking.No_of_people, booking.days, booking.breakfast,
booking.payment, booking.roomId,
receipt.Amount,customer.name AS customerName, customer.phoneNumber, customer.address,
customer.email, room.Roomtype FROM booking LEFT JOIN receipt ON booking.bookingId =
```

```
receipt.bookingId LEFT JOIN customer ON booking.customer = customer.id LEFT JOIN room ON
booking.roomId = room.Roomid WHERE booking.customer = 1002 ORDER BY booking.checkin DESC
```

8. Backend Implementation

8.1 Password Hashing

During Registration:

When a user registers on the platform, the password provided by the user (password) is hashed using the `bcrypt.hashSync()` function.

The hashed password is then stored in the database.

This process helps protect user passwords in case the database is compromised since only the hashed version of the password is stored.

During Login:

When a user logs in, their provided password (password) is compared against the hashed password stored in the database.

The `bcrypt.compare()` function is used to compare the plain-text password against the hashed password retrieved from the database.

If the comparison returns true, it means the passwords match, and the login is successful.

8.2 Commit and Rollback

Commit: The `connection.commit()` function is called after all database operations within a transaction are successfully completed. This commits the transaction, making all changes made within the transaction permanent in the database.

Rollback: The `connection.rollback()` function is called if any error occurs during the transaction or if the developer explicitly decides to abort the transaction. This rolls back the transaction, undoing all changes made within the transaction.

commit and rollback logic ensures that database operations during registration process which include serial insertion into customer or staff table and login table are atomic and consistent, maintaining data integrity even in the event of errors or failures during the transaction process. This helps prevent data corruption and ensures that the database remains in a consistent state.

8.3 Two-factor-authentication-using-speakeasy

When a new user signs up for an account, two-factor authentication (2FA) is implemented to enhance the security of the registration process. After the user provides basic account information such as name, email, and password, they are prompted to enable 2FA. During this setup a unique secret key is generated which is securely stored in the database. Additionally, the user is provided with a QR code containing this secret key, which is sent to the registered email using nodemailer library, which they can scan using a compatible authenticator app on their smartphone preferably Google Authenticator. This secret key is then encrypted using cryptographic techniques and stored in the database. When the user tries to log in, they must provide their username and password as the first factor of authentication. The system also prompts them to enter the second factor, typically a one-time password (OTP) generated by their authenticator app. The server retrieves the encrypted secret key from the database, decrypts it using the decryption method, and validates the OTP provided by the user. If both factors are successfully verified, the user is granted access to their account. This two-step verification process significantly strengthens the security of the authentication process, reducing the risk of unauthorized access and enhancing overall account protection.

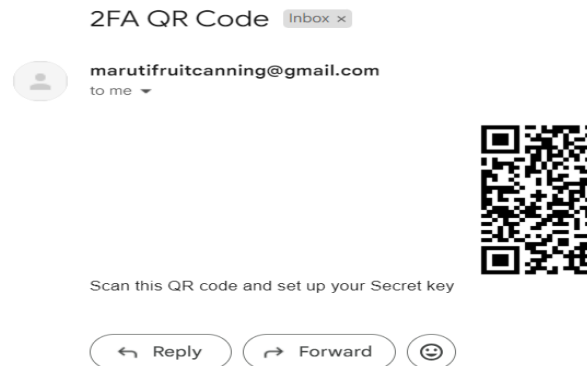


Fig 8.1: QR code sent to registered email

Speakeasy

Speakeasy is a versatile JavaScript library that facilitates the implementation of two-factor authentication (2FA) in web applications. One of its key features is the generation and verification of time-based one-time passwords (TOTPs). TOTPs are dynamic passcodes that change periodically and are valid only for a short duration, typically 30 seconds.

Speakeasy provides robust mechanisms for TOTP generation and validation, supporting various cryptographic algorithms and encoding formats. It also offers flexibility in configuring parameters such as the time window for accepting TOTPs and the length of the generated codes. By integrating Speakeasy into their applications, developers can enhance security by adding an additional layer of protection beyond traditional password-based authentication.

Crypto: Encrypting and Decrypting 2FA Secrets

When implementing two-factor authentication (2FA), the security of sensitive information like secret keys is paramount. To safeguard these keys, cryptographic techniques are employed. When a user sets up 2FA during registration, a unique secret key is generated for them. Before storing this key in the database, it undergoes encryption using cryptographic algorithms provided by libraries like Node.js's built-in crypto module. This process ensures that even if the database is compromised, the secret keys remain secure.

During login, the encrypted secret key is retrieved from the database and decrypted using the appropriate decryption method. Only authorized users possessing the correct credentials and OTPs can access the keys required for 2FA. This encryption-decryption mechanism plays a crucial role in maintaining the integrity and confidentiality of user data, contributing significantly to the robustness of the authentication process.

8.4 web services and API table

These APIs represent the endpoints of web services provided by the application. These endpoints define the routes that clients can access to interact with your application. Each endpoint corresponds to a specific functionality or resource, and clients can make HTTP requests to these endpoints to perform actions or retrieve data from your server. Clients, such as web browsers or mobile applications, can make HTTP requests to these endpoints to access the corresponding functionality provided by your application.

Table 8.1: Routes Table

Route	Method	Description
/	GET	Render the signup form
/signUp	GET	Render the signup form
/login	GET	Render the login form
/authenticate	GET	Render the 2FA authentication form
/home	GET	Render the home page
/profile	GET	Render the profile page
/success	GET	Render the success page
/error	GET	Render the error page
/staff	GET	Render the staff dashboard page
/customer	GET	Render the admin dashboard page
/amenities	GET	Render the amenities page
/admin	GET	Render the initial staff dashboard page

9. Test Cases

Table 9.1: Test Cases

Test Case Description	Test Data/Input	Expected Outcome	Type of Test
Users sign up with valid credentials	Username: user1, Email: user1@example.com , Password: 123456 (...other details)	User account is created successfully. Email verification link is sent.	Unit Test
Users sign up with existing email address	Username: user2, Email: user1@example.com , (...other details)	Sign up fails, displaying an error message indicating that the email address is already in use.	Unit Test
User login with valid credentials	Email: user1@example.com , Password: 123456, Secret Key: 345789	User successfully logs in and is redirected to the dashboard.	Unit Test
User login with incorrect password	Email: user1@example.com , Password: wrongpassword	Login fails, displaying an error message indicating incorrect password.	Unit Test
User login with non-existent email address	Email: non-existent@example.com , Password: 123456	Login fails, displaying an error message indicating that the email address does not exist.	Unit Test
User email verification	Click on the email verification link	User's email is verified successfully, and the account is activated.	Unit Test
User authentication with authenticator code	Authenticator code: 123456	User is authenticated successfully and logged in.	Unit Test
User booking a room for available dates	Room: Deluxe Room, Check-in: 2024-05-10, Check-out: 2024-05-15	Room is booked successfully for the specified dates.	Unit Test
User booking a room for already booked dates	Room: Deluxe Room, Check-in: 2024-05-12, Check-out: 2024-05-16	Booking fails, displaying an error message indicating that the room is already booked for some or all of the specified dates.	Unit Test
User viewing past bookings	Click on "Past Bookings" button	User's past bookings are displayed on the screen.	Unit Test
User navigating to profile page	Click on "Profile" button	User's profile page is displayed with the user's information.	Unit Test

Successful login redirects staff to separate dashboard	Valid login credentials	Staff is redirected to the staff dashboard	Functional
View all reviews button functionality	Clicking "View all reviews" button on the dashboard	All reviews are displayed for the staff to view	Functional
View all amenities button functionality	Clicking "View all amenities" button on the dashboard	All amenities are displayed for the staff to view	Functional
Add a service functionality	Customer ID, room ID, service type	Service details are saved to the database	Functional
Add amenity functionality	Amenity details	New amenity is added to the database	Functional
Customize amenities for each room functionality	Select room, add or delete amenity from dropdown	Amenity is added or deleted for the selected room	Functional
Search customer by email and view booking history functionality	Customer email	Customer booking history is displayed	Functional

10. Cloud Database

10.1 Plan for future database maintenance and upgrades:

Regular Backups:

- Backing up your MySQL database regularly is essential for data safety and disaster recovery. Use Amazon RDS automated backups or third-party tools to schedule regular backups. Automated backups ensure that your data is consistently backed up without manual intervention.
- Additionally, consider taking manual snapshots before making significant changes to the database, such as schema modifications or version upgrades. Manual snapshots provide a point-in-time recovery option and can be useful for rolling back changes if necessary.

Monitoring and Performance Tuning:

- By monitoring the performance of your MySQL database we can determine any issues that may creep in, that will eventually affect the users. Amazon RDS offers the provision of monitoring capabilities that will enable you to gather statistics such as CPU utilization, storage space, and query duration.

- Notice the critical thresholds with the alerts, to resolve proactively the performance issues. To explain, you will be notified by sending a message when CPU utilization will be high or when the storage is running out.
- It is essential to check and improve database queries and indexes and to do so regularly to increase performance. With the MySQL query profiler as a tool, locate slow queries and improve their efficiency with better optimizations.

Security Measures:

- Security of MySQL database should be top-priority because sensitive information is at stake. Keep security patches and updates for MySQL database engine and the underlying operating system updated to fix any potential vulnerabilities.
- Review the access and permission of users constantly to make sure they have the minimal privileges needed for their roles. Follow the principle of least privilege to ensure the least possibility of unauthorized access.
- Security tools like VPCs, security groups and encryption at rest and in transit should be used to prevent intrusion and interception of data.

Version Upgrades:

- Keeping both MySQL and Node. running. It is necessary to keep js versions up to date because such issue enhances security safety, performance, and the usage of latest features and improvements. Keep yourself updated on the new launches and version improvements and plan them appropriately.
- Firstly, we should patch the lower versions in a stage environment to find what might be the issues with the compatibility of the newly upgraded versions to the existing code or configuration. Testing makes sure the better way of doing things is succesful without messes or bumps.

Schema Changes:

- Database schema changes require a lot of caution in preparation and execution in order for data loss or corruption to be avoided. Along with this, document all the schema changes and migrations so that all database changes over time are captured on a record.
- Make use of database schema change management tools like Liquibase or Flyway which systematically take care of the required changes. Tracking and managing changes to a system becomes less complicated with version control, so that a reuse of prior version is possible when needed.

- Being cautious before making schema changes in a production environment is important. Try to use transactions and back up data as much as you can because these measures can help to prevent unintended consequences.

Database Replication and Failover:

- Implementing database replication and failover mechanisms ensures high availability and fault tolerance for your MySQL database. Multi-AZ deployment in Amazon RDS provides automatic failover in case of instance failure, improving overall system reliability.
- Consider setting up read replicas to offload read-heavy workloads and improve scalability. Read replicas can also serve as failover targets in case the primary instance becomes unavailable.

Regular Testing:

- The process of testing the hotel management system is actually of significant weight as far as proving the system's performance, functionality and scalability are concerned. Put forward a set of rigorous tests which will deal with both the database and application layers so that their performance can be monitored uninterruptedly.
- Draw various load and performance tests that imitate real world usage scenarios and thereby detect bottlenecks or scalability problems. Testing detects flaws earlier enough before it compromises the users' experience and as such, it enables the implementation of little changes and resolution issues that may arise after the product has been launched.

Documentation and Knowledge Sharing:

- The preservation of all the details of the database schema, settings, and maintenance procedures as a vital part of knowledge sharing and sustaining the system is critical. We are creating data diagrams and documenting best practices. This assists us with onboarding new team members and also it helps us to troubleshoot and resolve problems.
- Inculcate a learning culture by which team members share knowledge to make sure that everyone understands database architecture, operations, and procedures. Subscene shall recheck and update the document to ensure that it is updated to the current situation.

Disaster Recovery Planning:

- A disaster recovery plan is a must for avoiding possible downtime and data loss in case of a catastrophic breakdown. Outline process that entails rapid recovery of data and swift restoration of service.

- Run the disaster recovery plan frequently to confirm its efficiency and to spot any problems or deficiencies. This shows that planned testing is the basis of your organization's response to unforeseen events and minimizes the effect on operations.

10.2 Data privacy and security issues:

Encryption:

- All data stored in our RDS instances, both at rest and in transit, is encrypted using AWS Key Management Service (KMS) to ensure data privacy and security.
- We employ SSL/TLS encryption for securing data transmitted between our application and the RDS instance, safeguarding it from interception.

Access Control:

- Access to our RDS instances is tightly controlled following the principle of least privilege. Only authorized personnel and services are granted access.
- IAM roles and policies are utilized to manage access permissions, ensuring that each role has only the necessary privileges required for its specific tasks.

Secrets Management:

- Sensitive information such as database credentials are securely managed using AWS Secrets Manager or AWS Systems Manager Parameter Store.
- Credentials are never hardcoded in our application code or configuration files, instead, they are dynamically retrieved from the secrets store during runtime.

Network Security:

- Our RDS instances are deployed in a private subnet with carefully configured network security groups to regulate inbound and outbound traffic effectively.
- AWS VPC endpoints are utilized to enable secure communication between our application and the RDS instance, minimizing exposure to the public internet.

Audit Logging and Monitoring:

- Logging features such as Amazon CloudWatch Logs and AWS CloudTrail are enabled to track database events and API activity, providing comprehensive audit trails for security analysis.
- We have set up robust monitoring and alerting mechanisms to detect and respond to any anomalous database activity or security incidents promptly.

Disaster Recovery:

- Our multi-AZ deployment strategy ensures high availability and disaster recovery. In the event of a failure in one availability zone, AWS automatically fails over to a standby instance in another zone.
- We conduct regular tests of our disaster recovery procedures to validate their effectiveness and ensure minimal downtime in case of a catastrophic failure.

Secure Deployment Pipeline:

- Our pipeline deploys to AWS with strong security parameters in order to secure sensitive data and make sure that the communication with the AWS services is safe.
- Encryption and strong access controls for all sensitive data deployed on production including stored within secrets such databases credentials are among other places used.

Regular Security Audits:

- consistently perform internal security audits and assessments of our AWS systems and web applications, enabling us to spot and eliminate any potential security risks.
- Applying timely patches and updates is one of the ways to cure security breakdowns and keep to the best practices of security means from AWS.

Testing Backup and Recovery Procedures:**Purpose:**

- The purpose of testing backup and recovery procedures is to ensure the integrity and reliability of our data backups and the effectiveness of our recovery processes in case of data loss or system failure.

Backup Strategy:

- We utilize AWS RDS automated backups, which are taken daily, to create point-in-time snapshots of our MySQL database.
- Additionally, manual snapshots are periodically created before performing significant database changes or upgrades to ensure a recovery point in case of unexpected issues.
- Backup snapshots are retained for a period of 10 days, providing a window for data recovery in case of data loss or corruption.

Testing Procedure:**Step 1: Trigger Backup:**

- Manually trigger a backup of the MySQL database using the AWS RDS console or CLI to simulate a daily backup operation.

Step 2: Data Verification:

- After the backup is completed, verify the integrity of the backup data by inspecting the snapshot details in the AWS Management Console.

Step 3: Recovery Simulation:

- Simulate a data loss scenario by intentionally deleting or corrupting a portion of the database.
- Initiate the recovery process by restoring the database from the most recent automated backup.

Step 4: Data Validation:

- Once the recovery process is completed, validate the restored data to ensure that it matches the state of the database before the simulated data loss.

Results and Observations:

- Recording any findings or challenges during the testing example, the time taken for backup and recovery processes.
- In case, it is applicable, mark the areas where the original data differs from the restored one and examine the reason for any discrepancies.

10.3 Performance and Scalability Evaluation:

Performance Evaluation:

- **Monitoring Metrics:** Monitor the most important metrics like CPU utilization, memory usage, disk I/O and query execution times using Amazon CloudWatch or similar tools.
- **Query Optimization:** Analyse and optimize the slow running queries using mysql's query profiler or any other similar tool to identify and tackle the bottlenecks.
- **Indexing Strategy:** Provide effective data retrieval by enhancing indexes performance and conducting periodic review and maintenance.
- **Configuration Tuning:** Configure database settings for buffer pool size, cache sizes, and other similar parameters to attain better performance under a variety of workloads.

Scalability Evaluation:

- **Load Testing:** Carry out test for load to imitate actual usage scenarios and estimate the database's performance when traffic is either higher or lower than the realistic scenarios.
- **Performance Monitoring:** Serve as the Performance Indicator during load testing and the database's capability to scale and cope with the increasing workload.
- **Scalability Options:** Try vertical scaling out (fortifying instance size) before going with horizontal scaling (including reads replicas or sharding) as your database and users begins to grow.
- **Auto-Scaling:** Auto-scale policies might be implemented in order to automatically change resources to the level of demand, thus making it elastic and cost effective.

Benchmarking and Real-world Testing:

Benchmarking: Perform the benchmark analysis of the database performance against industrial performance standards or well-established benchmarks, to see what must be done to improve the performance.

Real-world Scenarios: Evaluate database capabilities in terms of scalability and performance during real-world conditions especially high-traffic periods and computationally challenging operations, to establish that it performs well under the relative variations.

With comprehensive performance and scalability testing in addition to implementing optimization strategies, we can guarantee that our out database can accommodate the demands of our hotel management system now and in the future. The addition of auto-scaling functions additionally extends our capacity to be flexible and maximizes performance while keeping costs down.

10.4 Database Maintenance Plan:

Regular Monitoring and Performance Tuning:

Utilize Amazon CloudWatch or similar monitoring tools to track key performance metrics such as CPU utilization, memory usage, disk I/O, and query execution times.

Set up alerts for critical thresholds to proactively identify and address performance issues.

Conduct regular reviews of slow-running queries and optimize them using MySQL's query profiler or similar tools.

Review and optimize database configuration settings, including buffer pool size and cache sizes, to ensure optimal performance under varying workloads.

Backup and Recovery Procedures:

Utilize AWS RDS automated backups, taken daily, to create point-in-time snapshots of the MySQL database.

Additionally, perform manual snapshots periodically before significant database changes or upgrades.

Store backup snapshots for a retention period of 10 days to ensure sufficient coverage for data recovery.

Regularly test backup and recovery procedures to verify data integrity and the effectiveness of recovery processes in case of data loss or system failure.

Security Measures:

Implement encryption for data both at rest and in transit using AWS KMS and SSL/TLS encryption.

Follow the principle of least privilege for access control, granting only necessary permissions to users and services.

Utilize AWS Secrets Manager or AWS Systems Manager Parameter Store to securely manage sensitive information such as database credentials.

Regularly apply security patches and updates for MySQL and the underlying operating system to address vulnerabilities and ensure compliance with security best practices.

Version Upgrades and Schema Changes:

Collecting Regular updates about new releases and updates for MySQL and Node.js, and plan for version upgrades to take advantage of performance improvements and new features.

Document all database schema changes and migrations using version control tools like Liquibase or Flyway, and practice caution when making schema changes in a production environment.

Regular Testing:

Undertake a regular load testing and performance testing to mimic user situations and uncover places where some issues may occur or the system need to be scaled. By following this maintenance plan, we ensure the reliability, performance, and security of our database infrastructure for our hotel management system. Regular monitoring, testing, and continuous improvement efforts help us adapt to evolving requirements and maintain a high standard of data integrity and system availability.

11. Conclusion and Future work:

The Hotel Management System is an intuitive web application that simplifies the process of booking and managing rooms. It has a user-friendly interface and can be adapted to meet the needs of hotels of varying sizes. The system also has plans to include room service booking, and it has the potential for expansion to a hotel chain, which highlights its future development.

Looking ahead, there is a prospect of further growth with the planned integration of an employee management system. This will ensure comprehensive and efficient administration. Additionally, the system incorporates well-designed customer and staff dashboards. These dashboards serve as intuitive interfaces for customers to navigate through room booking processes, view past bookings, and interact with additional features. At the same time, staff members experience a user-friendly environment for managing bookings, accessing customer information, and overseeing services.

11.1 Use of NoSQL, cloud computing, and big data technologies:

By transitioning to a microservices architecture, different components of the system can be decoupled and managed independently, enabling easier scaling and maintenance. Here's how the application can be scaled using microservices:

Reservation Management Microservice:

This microservice can benefit from using a relational database like MySQL due to its need for transactional integrity and complex querying capabilities. It manages reservation data, including bookings, cancellations, and modifications.

Room Management Microservice:

This microservice can use a NoSQL database like MongoDB or DynamoDB. It handles room details, availability, and amenities. NoSQL databases are suitable for managing semi-structured data like room configurations, amenities, and availability schedules. Additionally, it can utilize big data technologies to analyze booking patterns and optimize room allocation.

User Authentication and Authorization Microservice:

Relational databases are typically used for storing user credentials and profile information due to their need for ACID compliance and strong consistency. Big data technologies can be employed to analyze access patterns and detect anomalies for security purposes.

Payment Processing Microservice:

Payment transaction data requires strong consistency and reliability, making a relational database like MySQL or PostgreSQL a suitable choice. However, big data technologies can be employed to analyze transactional data for fraud detection, trend analysis, and financial forecasting. For example, Apache Kafka can be used for real-time transaction processing, while Apache Spark or Hadoop can perform batch processing for analytics.

Notification Microservice:

This microservice may use a combination of databases depending on the data's nature. Relational databases can store notification templates and delivery status for auditing purposes. NoSQL databases can be used to store user preferences and notification history. Big data technologies like Apache Kafka or Apache Flink can be used for real-time processing of notification events and delivery status updates.

11.2 Use of Mobile Devices:

The endpoints provided by the application can indeed be accessed from any browser on a mobile device, just like they can be accessed from a desktop browser. This is because web services are designed to be platform-independent, allowing any device with internet access and a web browser to interact with them.

So, whether a user is accessing application from a desktop computer, a tablet, or a smartphone, they can simply open their web browser, navigate to the appropriate URL (e.g.,

<http://localhost:8080/login>), and interact with the provided endpoints to perform various actions or retrieve data from your server.

We will consider developing a dedicated mobile application in the future. This would help to leverage device-specific features and provide a smoother user experience tailored to mobile users.