Hotel Booking System

Prepared by

[Hadžera Zukanović]
[Adi Okerić]
[Ada Kuskunović]
[Faris Kajević]

Contents

Database Systems

Introduction: Overview of the project

The Hotel Booking System is a project whose aim was to create a database system that offers a straightforward way of managing hotel reservations and keeping track of availability, all while meeting the needs of clients. To achieve said goal a database and website were designed and implemented as a solution that enables hotels to efficiently handle reservations.

Project Details

The process of creating a database can be shown through the database lifecycle, which follows a series of distinct steps designed to guarantee a well-designed database system. The steps are as follows:

1. The database initial study

1.1 Analyzing the company situation

When analyzing the company situation and providing a brief overview of a hotel booking system, the requirements and context of the organization were considered. This system needs to meet the needs of this hotel. The Oasis Bliss Resort is a new hotel with 15 available rooms. The needs of the hotel include allowing users to book rooms, keeping track of data about said users, being able to handle data about rooms and employees, running queries to find needed information, and keeping all the information up to date.

1.2 Defining problems and constraints

By identifying and addressing problems and constraints early (such as properly identifying the needs of the booking system, predicting functionalities to be implemented, and data integrity and security) in the development process, effective strategies and solutions were devised to ensure the successful implementation and operation of the hotel booking system. All the problems mentioned will be explained in more detail later in the report.

1.3 Defining objectives

The initial objective of the project is to provide a straightforward system that will allow hotel employees to manage bookings and guests. As this is the first system the hotel in question will use the goal is to make it as ideal as possible but still allow for future improvements if necessary. The system will share data with other systems the hotel uses such as the payment system.

1.4 Defining scope and boundaries

The scope would include designing a user-friendly interface for customers, creating features that track the availability of rooms, and allowing employees to manage user accounts and update/delete records on the website.

The hotel booking system operates within the following boundaries:

- The system will focus on managing reservations and not other hotel management aspects such as housekeeping or others.
- The system will ensure the security and privacy of customer information; however, it is not responsible for factors such as user negligence or unauthorized access to user devices.

These defined scope and boundaries provide a clear understanding of the functionalities and limitations of the hotel booking system.

2. Database design

The second phase in the database life cycle includes creating a conceptual design, choosing a DMBS, and creating a logical and physical design.

2.1 Creating a conceptual design

Entity Relationship (ER) Modeling and Normalization

After analyzing the company and its operations the following business rules were defined: Each hotel can have many bookings and each booking belongs to one hotel. Each guest can have many bookings and each booking belongs to one guest. Each hotel has many rooms and each room belongs to one hotel. Each room can be of one type and each room type has many rooms. A room can be booked many times and a booking can contain many rooms.

We began by creating a design for our database structure. We made a list of all the necessary tables and sketched out an Entity Relationship Diagram on paper. Each table was carefully designed with its respective columns and attributes, taking into consideration the relationships between the tables. Our goal was to minimize data redundancy. The final ER diagram that we created can be seen in Photo 1. Our team has implemented normalization in the database, ensuring that it meets the standards of the third normal form.

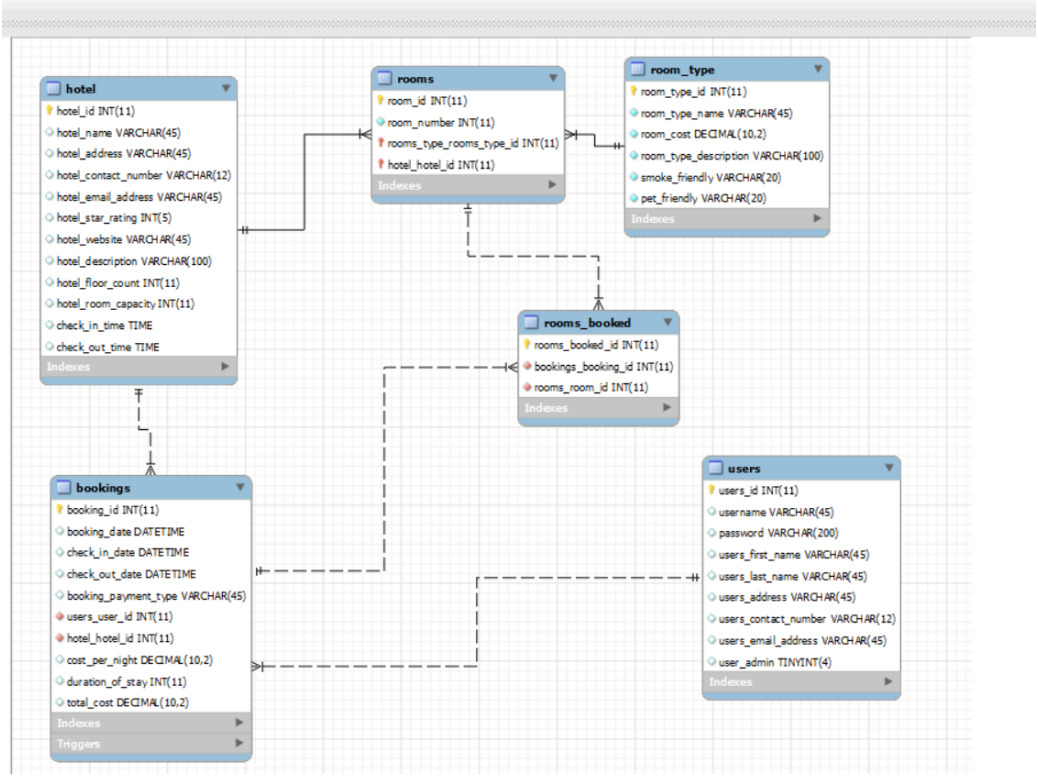


Photo 1 ERD

Data Model Verification

During the data model verification step, a thorough examination of the data model is conducted to identify any logical or structural inconsistencies that may exist within the model. This process involves analyzing the relationships, attributes, constraints, and dependencies defined in the data model to ensure they align with the intended requirements and business rules. By identifying and resolving these inconsistencies, the data model can be validated for its accuracy, completeness, and internal coherence, thereby enhancing the reliability and effectiveness of the overall database system.

Distributed database design

During the data model verification step, a thorough examination of the data model is conducted to identify any logical or structural inconsistencies that may exist within the model. This process involves analyzing the relationships, attributes, constraints, and dependencies defined in the data model to ensure they align with the intended requirements and business rules. By identifying and resolving these inconsistencies, the data model can be validated for its accuracy, completeness, and internal coherence, thereby enhancing the reliability and effectiveness of the overall database system.

2.2 DBMS Software Selection

MySQL Workbench was chosen as the database management system (DBMS) due to its powerful features and user-friendly interface. One advantage of MySQL Workbench over Oracle is its ease of use. However, Oracle offers more advanced features and scalability options, making it a preferred choice for enterprise-level applications.

2.3 Logical Design

The logical design phase consists of four steps: mapping the conceptual model to logical model components, validating the logical model using normalization, validating the logical model integrity constraints, validating the logical model against user requirements. All tables, their attributes and relationships between them were defined correctly and normalization to the third form was performed.

2.4 Physical Design

The physical design phase involves transforming the logical database design into a physical implementation. It includes decisions on indexing strategy; these are where indexes could be beneficial:

1. Table: room_type
   - An index on the "room_type_name" column could be useful for quick lookups and searches based on room type names.

2. Table: rooms
   - An index on the "room_number" column could be helpful for efficient retrieval of rooms based on room numbers.

3. Table: bookings
   - An index on the "check_in_date" and “check_out_date” columns would improve the performance of queries that involve searching or filtering bookings based on the check-in date and check-out date.
   - Another index on the "users_user_id" column would enhance the retrieval of bookings associated with specific users.

4. Table: users
   - An index on the "username" column would aid in fast retrieval of user information based on their usernames.

3. Implementation and loading

Install the DMBS, create the database, load or convert the data.

In this phase, the DBMS was installed and configured and the creation of the database began. The implementation phase involved defining the tables, their attributes, data types, and relationships, and

ensuring data integrity through the use of constraints and validations. Once the tables were created data was loaded into them as can be seen in Photo 2 below.

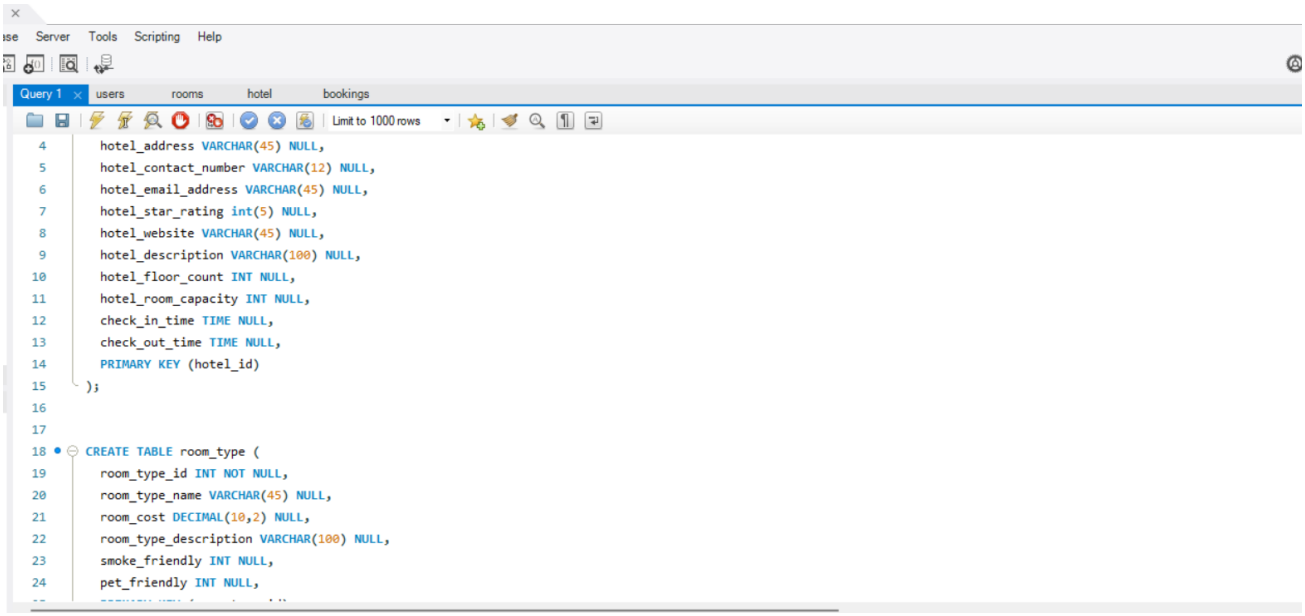


Photo 2 Creating tables

Photo 3 Inserting data into tables

Once the tables were created the process of writing queries began. The hotel booking system can generate queries to retrieve available rooms based on criteria such as date range, room type, and others enabling users to find suitable accommodations. Users can utilize the system to query booking details, including guest information, check-in/check-out dates, allowing them to modify existing reservations or inquire about additional services. All queries can be found in Appendix A.

Triggers

To further enhance the functionality of our database, we created triggers. All trigers, procedures and views can be found in Appendix 2.

4. Testing and evaluation

During this phase, a series of tests and evaluations are performed to ensure proper functioning of the system, its performance and its quality. Testing involves testing queries, procedures, correct data retrieval and updates. Photo 4 shows the correct use of a trigger that checks for available rooms.

Testing triggers

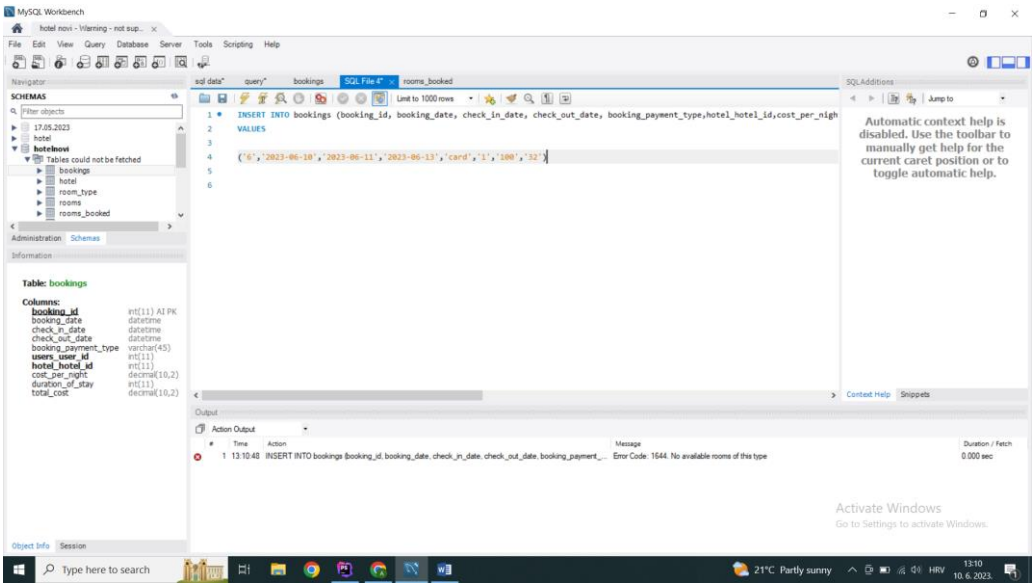


Photo 4 A trigger working properly

Testing queries

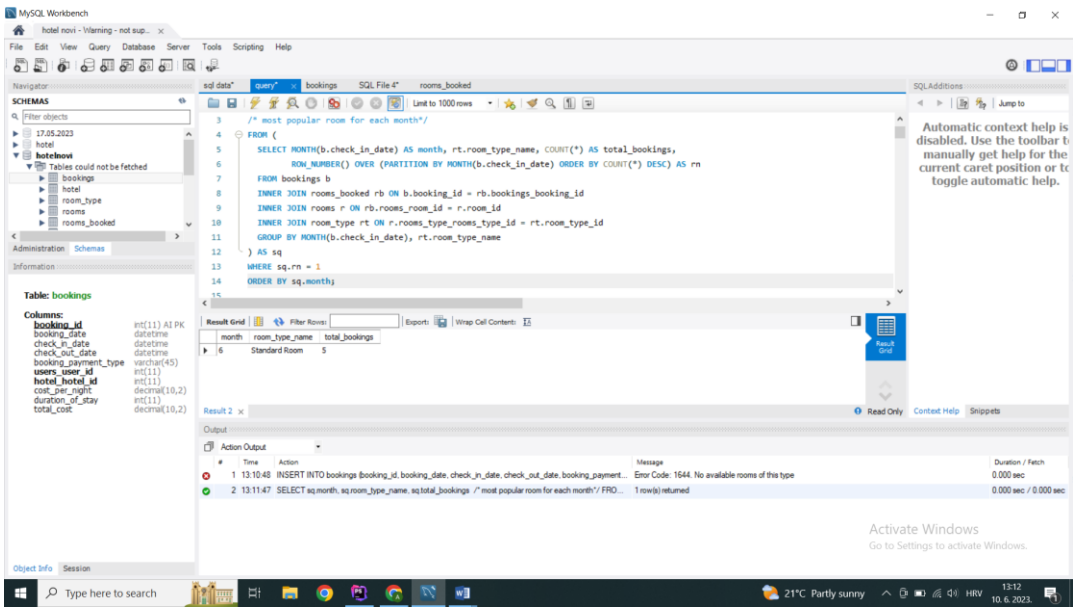


Photo 5 A query being tested

Photo 5 depicts the result set of a query that lists the most popular room for each month.

Successful delete and update operations

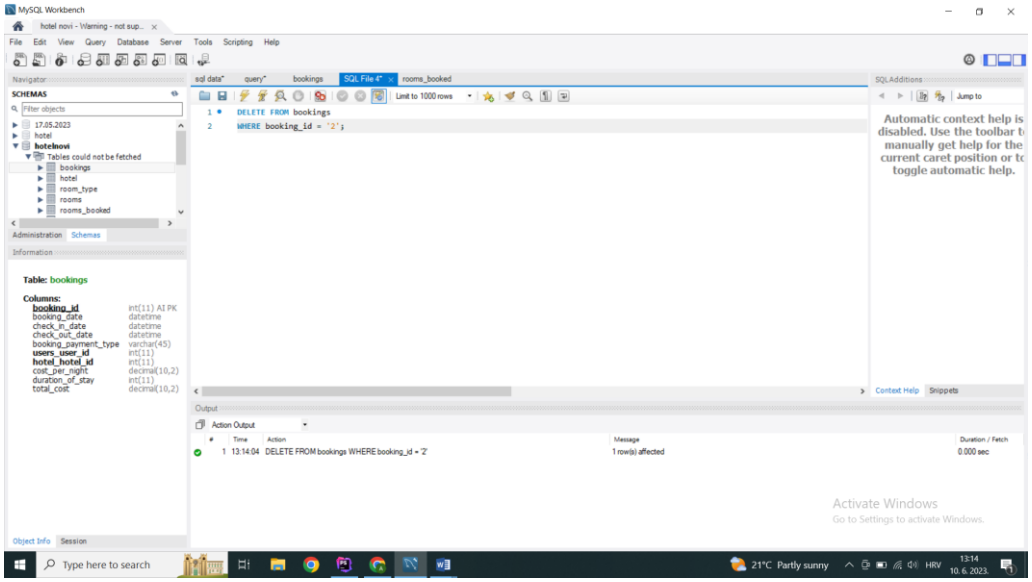


Photo 6 Delete operation tested

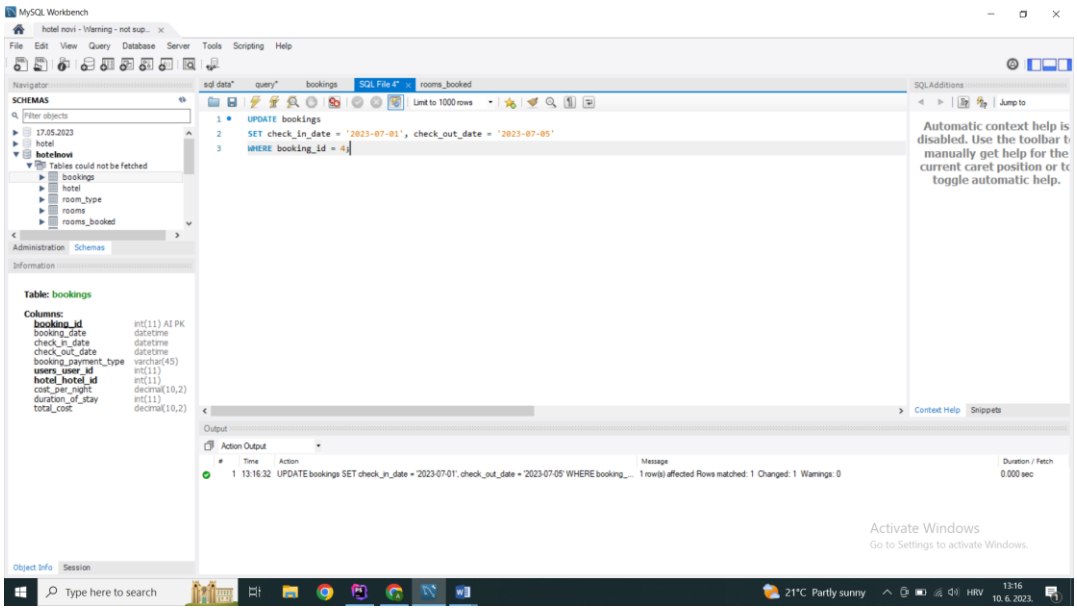


Photo 7 Update operation tested

5. Operation

The operation stage in the database life cycle refers to the ongoing usage and management of the database system. In this stage, the database is actively utilized by users to perform various tasks, including data entry, retrieval, modification, and reporting. The operation stage involves both regular operational activities and the administration of the database system. The hotel database has been undergoing user and admin usage completing all of the proposed tasks. Task handling, data manipulation, and error handling were constantly monitored during this stage to ensure flawless functionality of the database.

Overall, the operation stage focuses on ensuring the smooth and efficient operation of the database system.

6. Maintenance and evolution

When trying to ensure the functionality and adaptability of a database system it is important to focus on its maintenance and evolution. Monitoring the operations of the database and eliminating errors and threats ensures that integrity is kept.

Evolution plays an important role in a database. The ability of a database to adapt to the new needs of growing businesses reduces time consumption and cost. The hotel management system is designed in a totally flexible way suitable for adding new updates. For example, if this hotel becomes part of a

hotel chain and this system continues to be used adding new hotels is a very simple process. All of the tables in the database can be expanded in the future to meet the needs of the business.

Possible updates for the future

As mentioned in the last phase of the DBLC, this database can support many possible updates the hotel could make. Some of these updates include adding other hotels, creating new room types, or adding more rooms to the hotel. If the hotel decides to start offering other services those can be implemented into the current database with ease.

In conclusion, when taking into consideration that this is a new relatively small hotel this database should present no issues to its users in terms of usage and reliability.

To write this report the class book was used [1].

Appendix

Appendix A Queries

```
1. -- How many distinct guest have made bookings for a particular
month?
        SELECT users_first_name, users_last_name,users_contact_number
        FROM users
        WHERE users_id IN
                ( SELECT distinct users_user_id
                  FROM bookings
                  WHERE MONTH(check_in_date) = 6);


2.-- How many bookings has a customer made in one year?
        SELECT count(*) AS 'Total Bookings'
        FROM bookings
        WHERE YEAR(booking_date) = 2023 AND users_user_id = 32;


3. - How many rooms are booked in a particular hotel on a given date?
        SELECT COUNT(*) AS 'Total Rooms Booked'
        FROM bookings b
        INNER JOIN rooms_booked rb ON b.booking_id =
rb.bookings_booking_id
        WHERE b.booking_date = '2023-06-10'
  AND b.hotel_hotel_id = 1;




4. -- How many rooms are available in a given hotel?
         SELECT h.hotel_room_capacity - COUNT(rb.rooms_room_id) AS
'Available Rooms'
        FROM hotel h
        LEFT JOIN rooms r ON h.hotel_id = r.hotel_hotel_id
        LEFT JOIN rooms_booked rb ON r.room_id = rb.rooms_room_id
        WHERE h.hotel_id = 1;



5. -- List all the hotels that have a URL available.
        SELECT *
        FROM  hotel
        WHERE hotel_website IS NOT NULL;
```

6. -- Retrieve top 5 users who have booked the most rooms

```
        SELECT u.users_id, u.username, u.users_first_name,
u.users_last_name, COUNT(rb.rooms_booked_id) AS total_rooms_booked
        FROM users u
        JOIN bookings b ON u.users_id = b.users_user_id
        JOIN rooms_booked rb ON b.booking_id = rb.bookings_booking_id
        GROUP BY u.users_id, u.username, u.users_first_name,
u.users_last_name
        ORDER BY total_rooms_booked DESC
        LIMIT 5;
```

7. -- Retrieve the top 5 users who have spent the most amount of money

```
        SELECT u.users_id, u.username, u.users_first_name,
u.users_last_name, SUM(b.total_cost) AS total_amount_spent
        FROM users u
        JOIN bookings b ON u.users_id= b.users_user_id
        GROUP BY u.users_id, u.username, u.users_first_name,
u.users_last_name
        order by total_amount_spent DESC
        LIMIT 5;
```

8. -- Favorite room booked by guests along with all the information about the room

```
SELECT r.room_id, r.room_number, rt.room_type_name,rt.room_cost,
rt.room_type_description, rt.smoke_friendly, rt.pet_friendly
FROM rooms r
JOIN room_type rt  ON r.rooms_type_rooms_type_id= rt.room_type_id
JOIN rooms_booked rb ON r.room_id = rb.rooms_room_id
GROUP BY r.room_id, r.room_number, rt.room_type_name, rt.room_cost,
rt.room_type_description, rt.smoke_friendly, rt.pet_friendly
ORDER BY COUNT(rb.rooms_booked_id) DESC
LIMIT 1;
```

9. -- How much income for a certain hotel. Show the hotel name and the total booked amount by the hotel.

```
        SELECT h.Hotel_Name, SUM(b.total_cost) AS total_booked_amount
        FROM bookings b
        INNER JOIN hotel h ON b.hotel_hotel_id = h.hotel_id
        WHERE h.hotel_id = 1
        GROUP BY h.Hotel_Name;
```

10. -- Search for a room based on a room type
```
        SELECT r.room_id, r.room_number, rt.room_type_name
        FROM rooms r
        INNER JOIN room_type rt ON r.rooms_type_rooms_type_id =
rt.room_type_id
        WHERE rt.room_type_name = 'Standard room';
```

11. -- To search for a room within a price range
```
        SELECT r.room_id, r.room_number, rt.room_type_name, rt.room_cost
        FROM rooms r
        INNER JOIN room_type rt ON r.rooms_type_rooms_type_id =
rt.room_type_id
        WHERE rt.room_cost >= '50' AND rt.room_cost <= '100';
```

12. -- available rooms within a specific price range and room type
```
SELECT r.room_id, r.room_number, rt.room_type_name, rt.room_cost
FROM rooms r
INNER JOIN room_type rt ON r.rooms_type_rooms_type_id = rt.room_type_id
WHERE rt.room_type_name = 'desired_room_type'
  AND rt.room_cost >= minimum_price AND rt.room_cost <= maximum_price
  AND r.room_id NOT IN (
    SELECT rb.rooms_room_id
    FROM rooms_booked rb
    INNER JOIN bookings b ON rb.bookings_booking_id = b.booking_id
    WHERE (b.check_in_date <= desired_check_out_date AND
b.check_out_date >= desired_check_in_date)
  );
```

13. -- To modify an existing reservation by requesting changes to the check-in/check-out dates, room type, or other booking details
```
UPDATE bookings
SET check_in_date = '2023-07-10',
    check_out_date = '2023-07-15',
    hotel_hotel_id = '1',
    users_user_id = '32',
    total_amount = '100.00'
WHERE booking_id = '35';
```

14. --Standard delete query

```
DELETE FROM bookings
WHERE booking_id = '1';
```

15. -- Query for retrieving the upcoming bookings for a specific guest:

```
SELECT t.users_first_name, t.users_last_name, b.booking_id,
b.check_in_date, b.check_out_date, h.hotel_name
FROM bookings b
INNER JOIN hotel h ON b.hotel_hotel_id = h.hotel_id
INNER JOIN users t ON b.users_user_id = t.users_id
WHERE b.check_in_date >= CURRENT_DATE();
```

16. --This query will retrieve all rows from the users table where the uppercase version of the users_first_name column starts with 'ADI'. The % wildcard matches any sequence of characters following 'ADI'.

```
SELECT *
FROM users
WHERE UPPER(users_first_name) LIKE 'ADI%';
```

17. -- Query that shows the room types that have been booked for a total cost greater than $500

```
SELECT rt.room_type_name, SUM(b.cost_per_night) as total_cost
FROM room_type rt
INNER JOIN rooms r ON rt.room_type_id = r.rooms_type_rooms_type_id
INNER JOIN rooms_booked rb ON r.room_id = rb.rooms_room_id
INNER JOIN bookings b ON rb.bookings_booking_id = b.booking_id
GROUP BY rt.room_type_name
HAVING total_cost > 500;
```

18. -- Show all users which have a contact number longer than 4 digits

```
SELECT users_id, users_first_name, users_last_name,
users_contact_number
FROM users
GROUP BY users_id, users_first_name, users_last_name,
users_contact_number
HAVING LENGTH(users_contact_number) > 4;
```

Appendix B Triggers

1. -- A trigger that creates a new row in rooms_booked table stating the booking id and the room id which belongs to the booking

```
CREATE DEFINER=`root`@`localhost` TRIGGER create_rooms_booked_trigger
AFTER INSERT ON bookings
FOR EACH ROW
BEGIN
    DECLARE room_id INT;

    SET room_id = (
        SELECT r.room_id
        FROM rooms r
        INNER JOIN room_type rt ON r.rooms_type_rooms_type_id =
rt.room_type_id
        WHERE rt.room_cost = NEW.cost_per_night
            AND r.room_id NOT IN (
                SELECT rb.rooms_room_id
                FROM rooms_booked rb
            )
        LIMIT 1
    );

    IF room_id IS NULL THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'No available rooms of this type';
    ELSE
        INSERT INTO rooms_booked (bookings_booking_id, rooms_room_id)
        VALUES (NEW.booking_id, room_id);
    END IF;

END
```

---The trigger also checks if there are any available rooms of this type left to book, if there are no more available a error statement is shown

2. -- Trigger that is called when a booking is deleted so that the row which the booking id is referencing to in table rooms_booked can be deleted

```
CREATE DEFINER=`root`@`localhost` TRIGGER delete_rooms_booked_trigger
AFTER DELETE ON bookings
FOR EACH ROW
BEGIN
    DELETE FROM rooms_booked
    WHERE bookings_booking_id = OLD.booking_id;
END
```

Appendix C Functions

18. --Function that shows us all available rooms for a certain date

```
CREATE DEFINER=`root`@`localhost` FUNCTION
`get_available_rooms2`(check_in_date DATE, check_out_date DATE) RETURNS
varchar(255) CHARSET utf8mb4 COLLATE utf8mb4_general_ci
BEGIN
    DECLARE available_rooms VARCHAR(255);

    SET available_rooms = (
        SELECT GROUP_CONCAT(CONCAT(rt.room_type_name, ' (', (
            SELECT COUNT(r.room_id)
            FROM rooms r
            WHERE r.rooms_type_rooms_type_id = rt.room_type_id
                AND r.room_id NOT IN (
                    SELECT rb.rooms_room_id
                    FROM rooms_booked rb
                    INNER JOIN bookings b ON rb.bookings_booking_id =
b.booking_id
                    WHERE (check_in_date >= b.check_in_date AND
check_in_date < b.check_out_date)
                        OR (check_out_date > b.check_in_date AND
check_out_date <= b.check_out_date)
                        OR (check_in_date <= b.check_in_date AND
check_out_date >= b.check_out_date)
                        OR (check_in_date = b.check_out_date AND
check_out_date = b.check_in_date)
                )
        ), ' available)') SEPARATOR ', ')
        FROM room_type rt
    );

    IF available_rooms IS NULL THEN
        SET available_rooms = 'No available rooms.';
    END IF;

    RETURN available_rooms;
END
```

2 -- Function that checks for a given check in date and check out date if a booking exists

```
DELIMITER //
```

```
CREATE FUNCTION `check_for_bookings`(check_in_date DATE,
check_out_date DATE) RETURNS varchar(255) CHARSET utf8mb4 COLLATE
utf8mb4_general_ci

BEGIN

    DECLARE booking_info VARCHAR(255);


    SET booking_info = (

        SELECT CONCAT('Booking ID: ', b.booking_id, ', Check-in
Date: ', b.check_in_date, ', Check-out Date: ', b.check_out_date)

        FROM bookings b

        WHERE (check_in_date >= b.check_in_date AND check_in_date <=
b.check_out_date)

            OR (check_out_date >= b.check_in_date AND check_out_date
<= b.check_out_date)

            OR (check_in_date <= b.check_in_date AND check_out_date
>= b.check_out_date)

            OR (check_in_date = b.check_out_date AND check_out_date
= b.check_in_date)

        LIMIT 1

    );


    IF booking_info IS NULL THEN

        SET booking_info = 'No bookings found.';

    END IF;


    RETURN booking_info;

END


DELIMITER ;


 SELECT  check_booking_availability('2023-06-06', '2023-06-10');
```

Appendix D Views

1. -- View that shows us all bookings done with the first and last names of the users, as well as the type of room they booked and the numbers of the rooms

```
CREATE VIEW bookings_view AS
SELECT b.booking_id, b.booking_date, b.check_in_date, b.check_out_date,
b.total_cost, u.users_first_name, u.users_last_name,
       rt.room_type_name, r.room_number
FROM bookings b
INNER JOIN users u ON b.users_user_id = u.users_id
INNER JOIN rooms_booked rb ON b.booking_id = rb.bookings_booking_id
INNER JOIN rooms r ON rb.rooms_room_id = r.room_id
INNER JOIN room_type rt ON r.rooms_type_rooms_type_id =
rt.room_type_id;

SELECT * FROM bookings_view;
```

Appendix E Procedures

1. - - PROCEDURE THAT FOR A INPUTED PHONE NUMBER SHOWS ALL BOOKINGS MADE BY THE PERSON THE PHONE NUMBER BELONGS TO

```
CREATE DEFINER=`root`@`localhost` PROCEDURE
`GetBookingDetailsByPhoneNo`(IN guest_contact_number nvarchar(20))
BEGIN
    SELECT b.booking_id, b.booking_date, b.duration_of_stay,
b.check_in_date, b.check_out_date, b.booking_payment_type,
b.total_cost,
           h.hotel_name, h.hotel_address, h.hotel_contact_number,
h.hotel_email_address,
           rt.room_type_name, rt.room_cost,
rt.room_type_description, u.users_first_name
    FROM bookings b
    INNER JOIN hotel h ON b.hotel_hotel_id = h.hotel_id
    INNER JOIN users u ON b.users_user_id = u.users_id
    INNER JOIN rooms_booked rb ON b.booking_id =
rb.bookings_booking_id
    INNER JOIN rooms r ON rb.rooms_room_id = r.room_id
    INNER JOIN room_type rt ON r.rooms_type_rooms_type_id =
rt.room_type_id
    WHERE u.users_contact_number = guest_contact_number;
END
```

Appendix F Data Dictionary

Table: room_type

Columns:

room_type_id: INT (Primary Key)

room_type_name: VARCHAR(255)

room_cost: DECIMAL(10, 2)

room_type_description: VARCHAR(255)

smoke_friendly: BOOLEAN

pet_friendly: BOOLEAN

Table: rooms

Columns:

room_id: INT (Primary Key)

room_number: VARCHAR(20)

rooms_type_rooms_type_id: INT (Foreign Key referencing room_type.room_type_id)

hotel_hotel_id: INT (Foreign Key referencing hotel.hotel_id)

Table: bookings

Columns:

booking_id: INT (Primary Key)

booking_date: DATE

duration_of_stay: INT

check_in_date: DATE

check_out_date: DATE

booking_payment_type: VARCHAR(255)

users_user_id: INT (Foreign Key referencing users.users_id)

hotel_hotel_id: INT (Foreign Key referencing hotel.hotel_id)

total_amount: DECIMAL(10, 2)

Table: hotel

Columns:

hotel_id: INT (Primary Key)

hotel_name: VARCHAR(255)

hotel_address: VARCHAR(255)

hotel_contact_number: VARCHAR(20)

hotel_email_address: VARCHAR(255)

hotel_star_rating: INT

hotel_website: VARCHAR(255)

hotel_description: VARCHAR(255)

hotel_floor_count: INT

hotel_room_capacity: INT

check_in_time: TIME

check_out_time: TIME

Table: rooms_booked

Columns:

rooms_booked_id: INT (Primary Key)

bookings_booking_id: INT (Foreign Key referencing bookings.booking_id)

rooms_room_id: INT (Foreign Key referencing rooms.room_id)

Table: users

Columns:

users_id: INT (Primary Key)

users_first_name: VARCHAR(255)

users_last_name: VARCHAR(255)

users_email_address: VARCHAR(255)

users_contact_number: VARCHAR(20)

username: VARCHAR(45)

password: VARCHAR(200)

user_admin: TINYINT(4)

Appendix G Creation of tables

```
CREATE TABLE  hotel (
  hotel_id INT NOT NULL,
  hotel_name VARCHAR(45) NULL,
  hotel_address VARCHAR(45) NULL,
  hotel_contact_number VARCHAR(12) NULL,
  hotel_email_address VARCHAR(45) NULL,
  hotel_star_rating int(5) NULL,
  hotel_website VARCHAR(45) NULL,
  hotel_description VARCHAR(100) NULL,
  hotel_floor_count INT NULL,
  hotel_room_capacity INT NULL,
  check_in_time TIME NULL,
  check_out_time TIME NULL,
  PRIMARY KEY (hotel_id)
);


CREATE TABLE room_type (
  room_type_id INT NOT NULL,
  room_type_name VARCHAR(45) NULL,
  room_cost DECIMAL(10,2) NULL,
  room_type_description VARCHAR(100) NULL,
  smoke_friendly INT NULL,
  pet_friendly INT NULL,
  PRIMARY KEY (room_type_id)
);



CREATE TABLE rooms (
  room_id INT NOT NULL,
```

```
  room_number INT NULL,
  rooms_type_rooms_type_id INT NOT NULL,
  hotel_hotel_id INT NOT NULL,
  PRIMARY KEY (room_id, rooms_type_rooms_type_id, hotel_hotel_id),
  CONSTRAINT fk_rooms_rooms_type1 FOREIGN KEY
(rooms_type_rooms_type_id) REFERENCES room_type (room_type_id),
  CONSTRAINT fk_rooms_hotel1 FOREIGN KEY (hotel_hotel_id) REFERENCES
hotel (hotel_id)

);



CREATE TABLE  users  (
   users_id  INT NOT NULL,
   users_first_name  VARCHAR(45) NULL,
   users_last_name  VARCHAR(45) NULL,
   users_address VARCHAR(45) NULL,
   users_contact_number  VARCHAR(12) NULL,
   users_email_address  VARCHAR(45) NULL,
   users_credit_card  VARCHAR(45) NULL,
   users_id_proof  VARCHAR(45) NULL,
  PRIMARY KEY ( users_id )

);



 CREATE TABLE bookings  (
   booking_id  INT NOT NULL,
   booking_date  DATETIME NULL,
   duration_of_stay  VARCHAR(10) NULL,
   check_in_date  DATETIME NULL,
   check_out_date  DATETIME NULL,
```

```
   booking_payment_type  VARCHAR(45) NULL,
   total_rooms_booked  INT NULL,
   hotel_hotel_id  INT NOT NULL,
   users_user_id  INT NOT NULL,
   total_amount  DECIMAL(10,2) NULL,
  PRIMARY KEY ( booking_id ),
  CONSTRAINT  fk_bookings_hotel1 FOREIGN KEY ( hotel_hotel_id )
REFERENCES   hotel  ( hotel_id ),
  CONSTRAINT fk_bookings_guest1 FOREIGN KEY (users_user_id)
REFERENCES users(users_id)


);



CREATE TABLE rooms_booked  (
   rooms_booked_id  INT NOT NULL,
   bookings_booking_id  INT NOT NULL,
   rooms_room_id  INT NOT NULL,
  PRIMARY KEY ( rooms_booked_id),
  CONSTRAINT  fk_rooms_booked_bookings1 FOREIGN KEY (
bookings_booking_id )REFERENCES  bookings  ( booking_id ) ,
  CONSTRAINT  fk_rooms_booked_rooms1 FOREIGN KEY ( rooms_room_id
)REFERENCES rooms  ( room_id )



);
```

Web Design and Development

Abstract

This report outlines the development process of a hotel administration and booking system using PHP, HTML, and CSS. The major goal of the project was to develop an intuitive website where hotel managers could easily manage their business and visitors could book rooms. To create an eye-catching design and enhance the website's appearance, we used HTML and CSS. Using PHP, login and signup functionality was added, enabling users to create accounts and log in safely. Importantly, the project is connected to a database, which serves as a storage and retrieval system for data related to hotel activities and room reservations. The resultant website makes it simple to monitor hotel activities and reserve rooms without bother for both hotel personnel and visitors. This project serves as an example of how PHP, HTML, and CSS are used in the real world to create dynamic online apps for hotel administration.

Introduction

Working with the above-mentioned database the goal was to create a comprehensive website that will allow users to browse available rooms, view details for each room type, and make online bookings with just a few clicks. The Oasis Bliss Resort is a new hotel and currently is not a part of any hotel chain. The hotel offers accommodation in 15 rooms, each room type having 5 rooms. Taking in consideration the size and needs of the hotel we began the process of creating the website.

Project Implementation

To provide an aesthetically pleasing and useful user experience, we began with the front-end implementation of the hotel booking website landing page. Our goal was to create an elegantly designed website that will captivate prospective guests. The design of the website involved the use of HTML, CSS, and PHP technologies. The website's elements were organized using HTML. The visual presentation of the website, including its layout, colors, font, and overall styling, was defined using CSS. As this is the first thing potential guests see when going to the website the goal was to make the interface as intuitive as possible, offering even unexperienced internet users easy navigation. Tutorials found on the following websites [2] and [3], helped us in creating our web page.

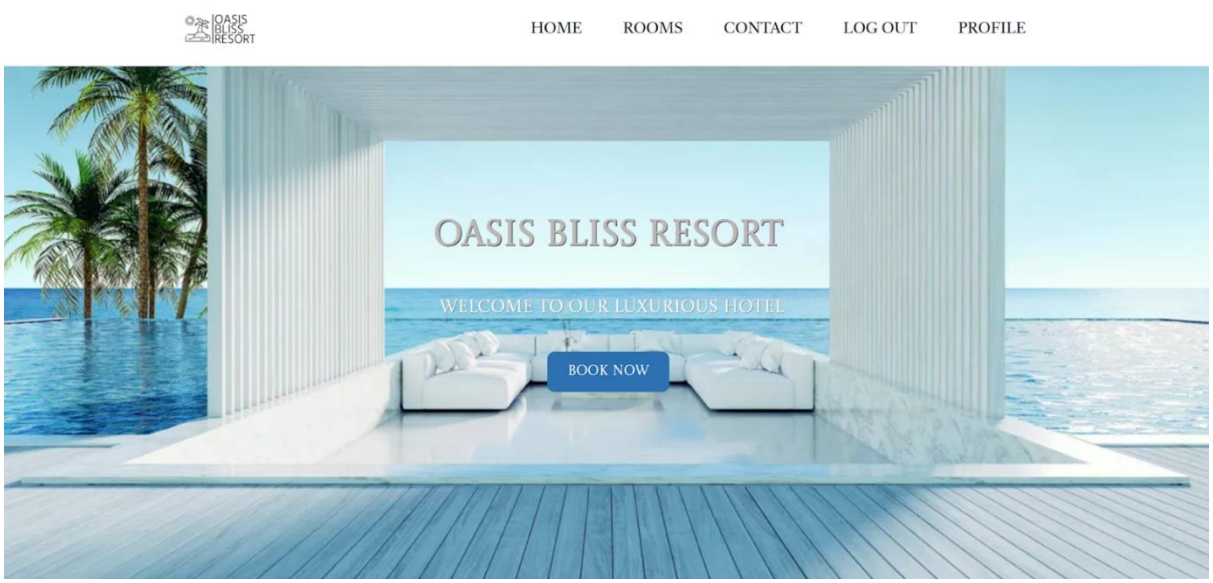


Photo 8 Home page

Users can effortlessly navigate through the available options on the navigation bar, which enable them to explore various room choices, create a new account, log in to their existing account, or view and manage their profile.

Upon clicking the sign-up button, users are redirected to a form that facilitates the creation of a new account. This intuitive process enables individuals to enter the required details and successfully register for an account, ensuring a hassle-free user experience.

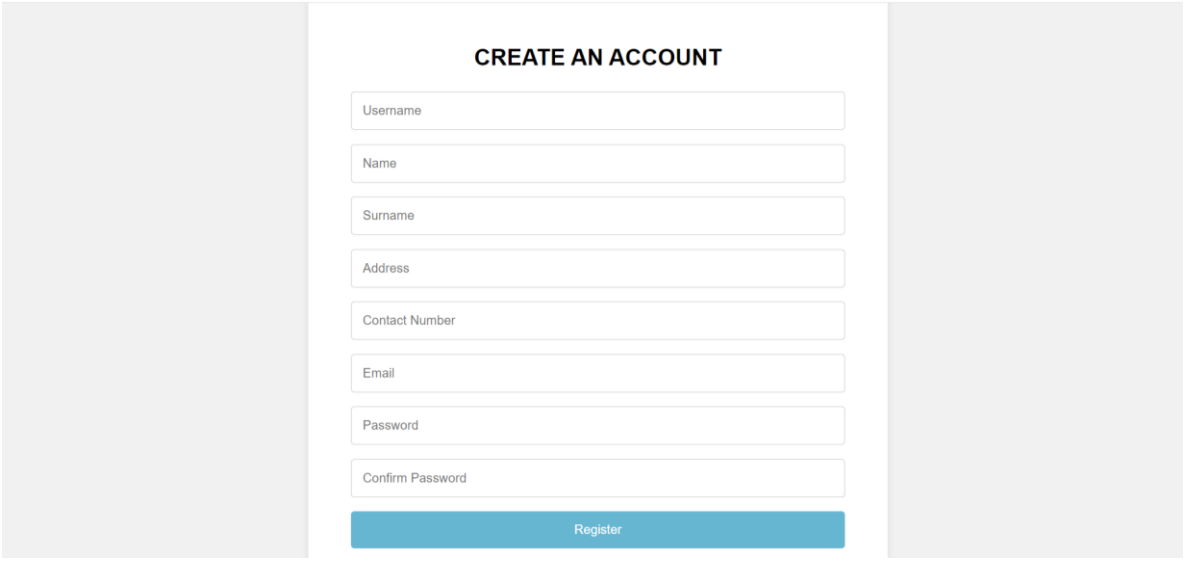


Photo 9 Sign up page

If users have previously created an account they can simply log in using their data.

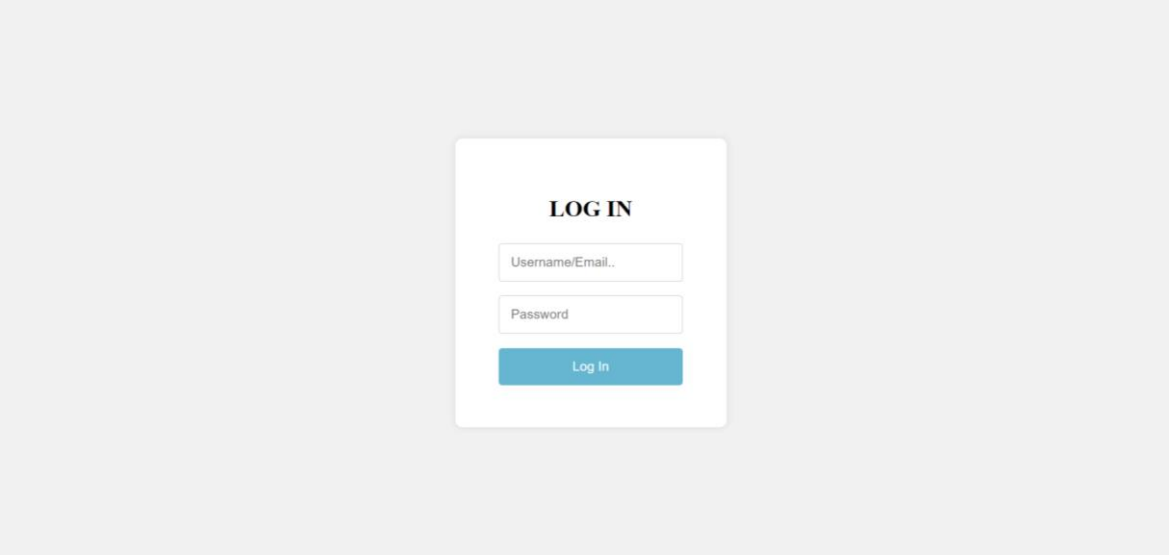


Photo 10 Log in page

Users can also view their own profile.

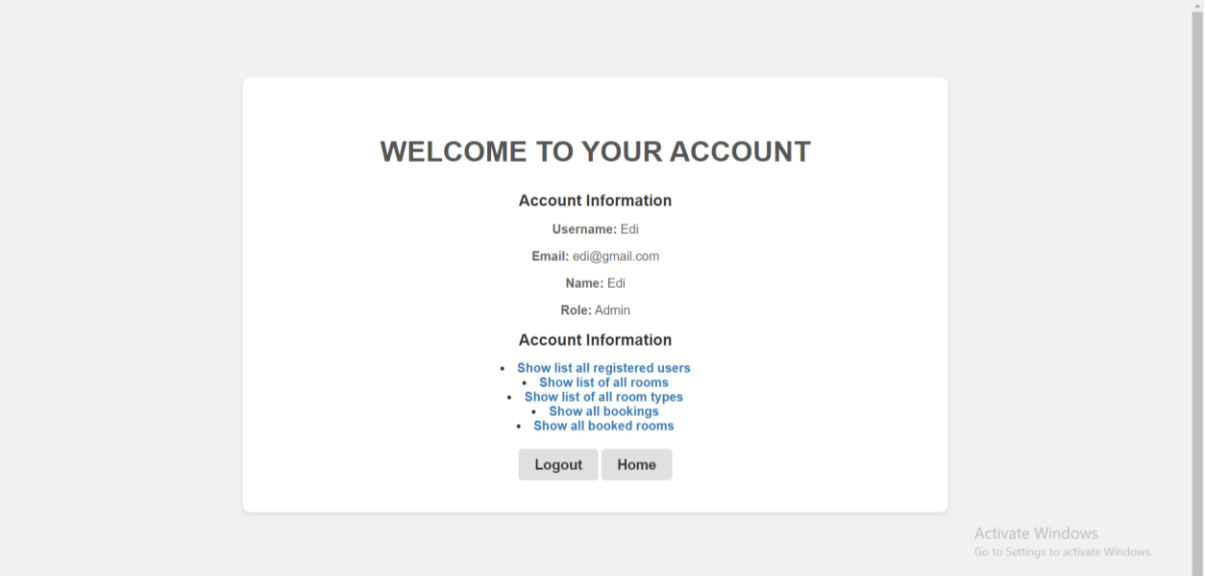


Photo 11 An employee's view of their profile

Guests have the liberty to browse through the variety of rooms available at the hotel and select the one that best meets their preferences. Once they have made a confident choice, they can simply click the "book" button, which will redirect them to a form that requires completion. Upon filling out the necessary information in the form, it is promptly submitted to the database, officially confirming the booking of the chosen room. This efficient process ensures a seamless transition from room selection to reservation confirmation.

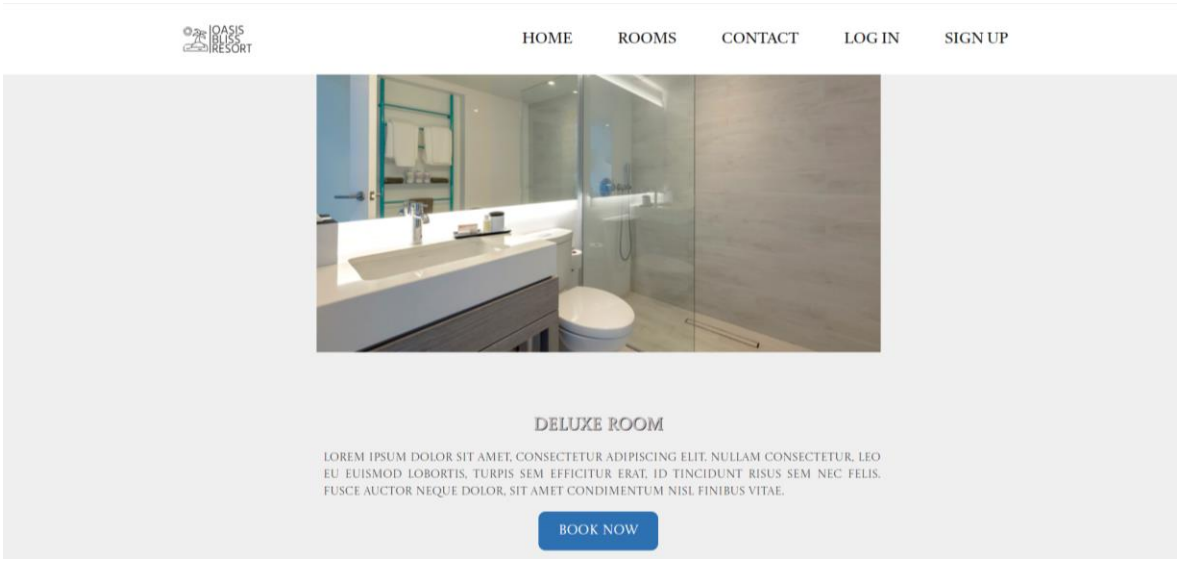


Photo 12 One of the rooms the hotel offers

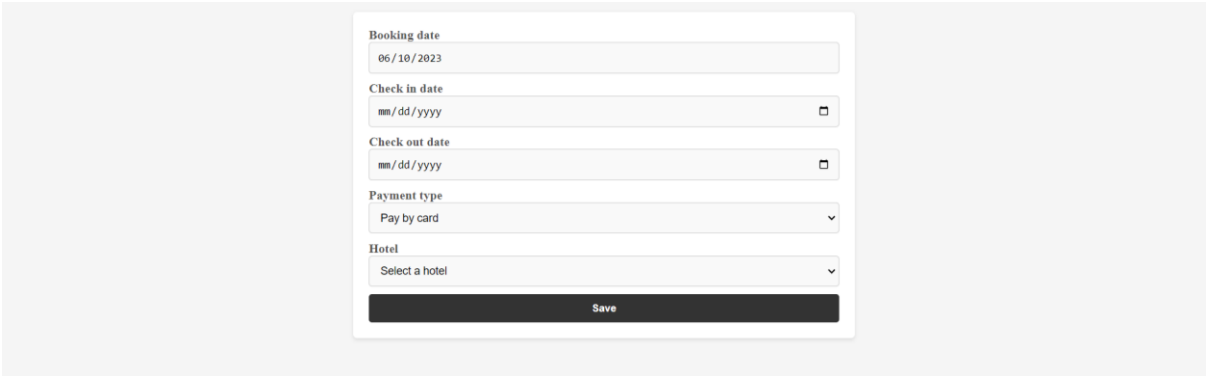
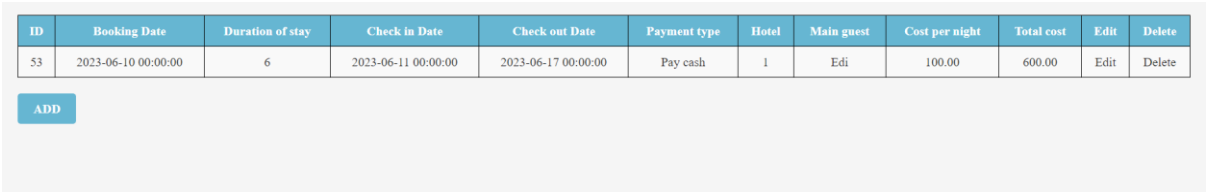


Photo 13 A form used to book a room



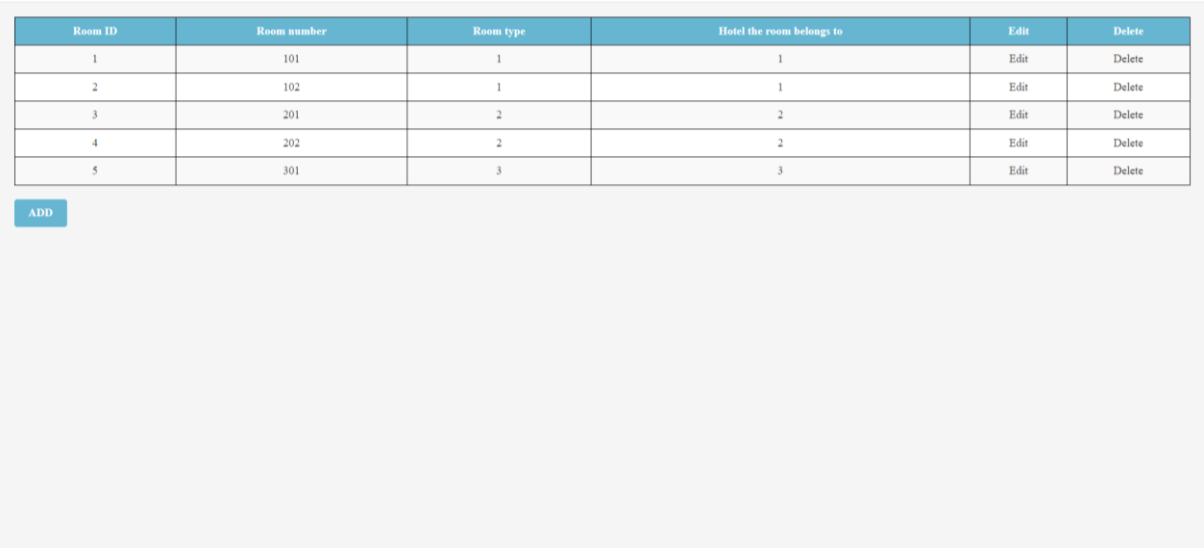
ID	Booking Date	Duration of stay	Check in Date	Check out Date	Payment type	Hotel	Main guest	Cost per night	Total cost	Edit	Delete
53	2023-06-10 00:00:00	6	2023-06-11 00:00:00	2023-06-17 00:00:00	Pay cash	1	Edi	100.00	600.00	Edit	Delete

ADD

Photo 14 Updated database after guests fill out the form and book a room

When an employee of the hotel accesses the website, they can effectively manage the database by utilizing the application's back-end functionalities. Through the back-end system, the employee can efficiently handle various database operations, such as updating, modifying, and deleting data. Our website utilizes sessions to grant administrative privileges, enabling authorized personnel to securely

access and modify confidential data. Below is an example of the list, delete, add and update operations that employees can perform.



Room ID	Room number	Room type	Hotel the room belongs to	Edit	Delete
1	101	1	1	Edit	Delete
2	102	1	1	Edit	Delete
3	201	2	2	Edit	Delete
4	202	2	2	Edit	Delete
5	301	3	3	Edit	Delete

ADD

Photo 15 Example of an employee's screen when viewing data about rooms. Here they can update or delete the data as well.

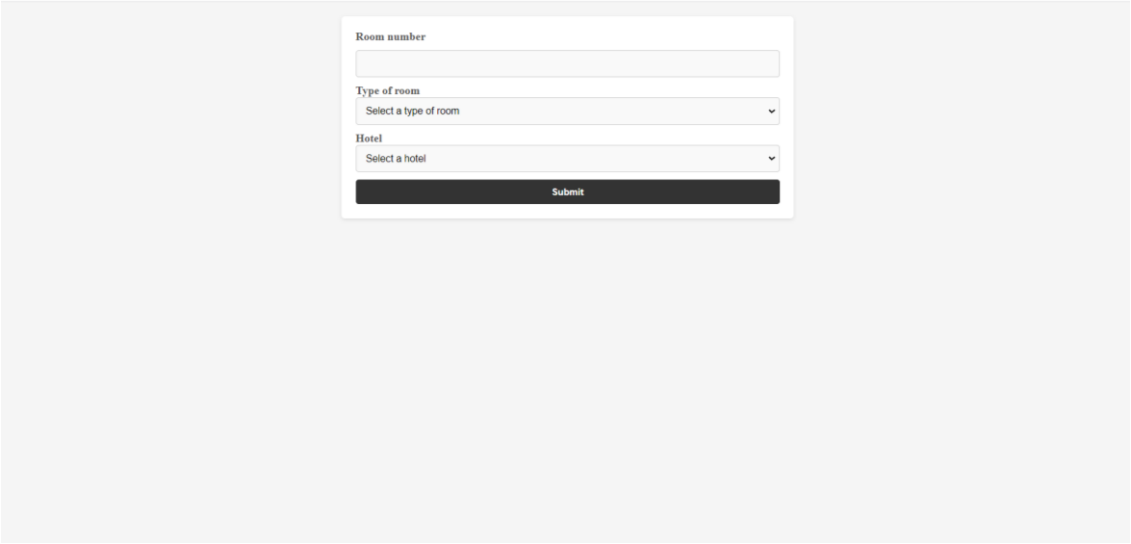


Photo 16 Example of an employee's screen when adding new data about a room

Lessons learned and conclusions

When we were creating this project our team acquired much knowledge in CSS, HTML, and PHP. Although we had prior experience with HTML and CSS, this project has broadened our horizons as this was our first time building a more complex online platform.

Our biggest lesson came from working with PHP during the creation of CRUD operations, as we had limited prior exposure to PHP. However, the process of integrating the database and designing the website proved instrumental in understanding the inner workings of modern websites. We gained insights into extracting data from databases and presenting it aesthetically using HTML and CSS.

When writing code in PHP we found help in our class book [4].

Bibliography

[1] Carlos, C. Coronel and S. Morris, Database Systems Design, Implementation & Management 13th Edtion, Boston: Cengage, 2020.

[2] "W3 SCHOOLS," [Online]. Available: https://www.w3schools.com/html/default.asp. [Accessed 30 May 2023].

[3] "W3SCHOOLS," [Online]. Available: https://www.w3schools.com/css/default.asp. [Accessed 31 May 2023].

[4] L. Ullman, PHP for the Web 5th Edition, San Francisco: Visual QuickStart Guide, 2016.

Table of figures