Hotel Management System

The project submitted to the

SRM University – AP, Andhra Pradesh

for the partial fulfillment of the requirements to award the degree of

Bachelor of Technology

In

Computer Science and Engineering

School of Engineering and Sciences

Submitted by

Namratha Addagada (AP21110010048)

Under the Guidance of

Dr. Ch. Anil Carie

Assistant Professor

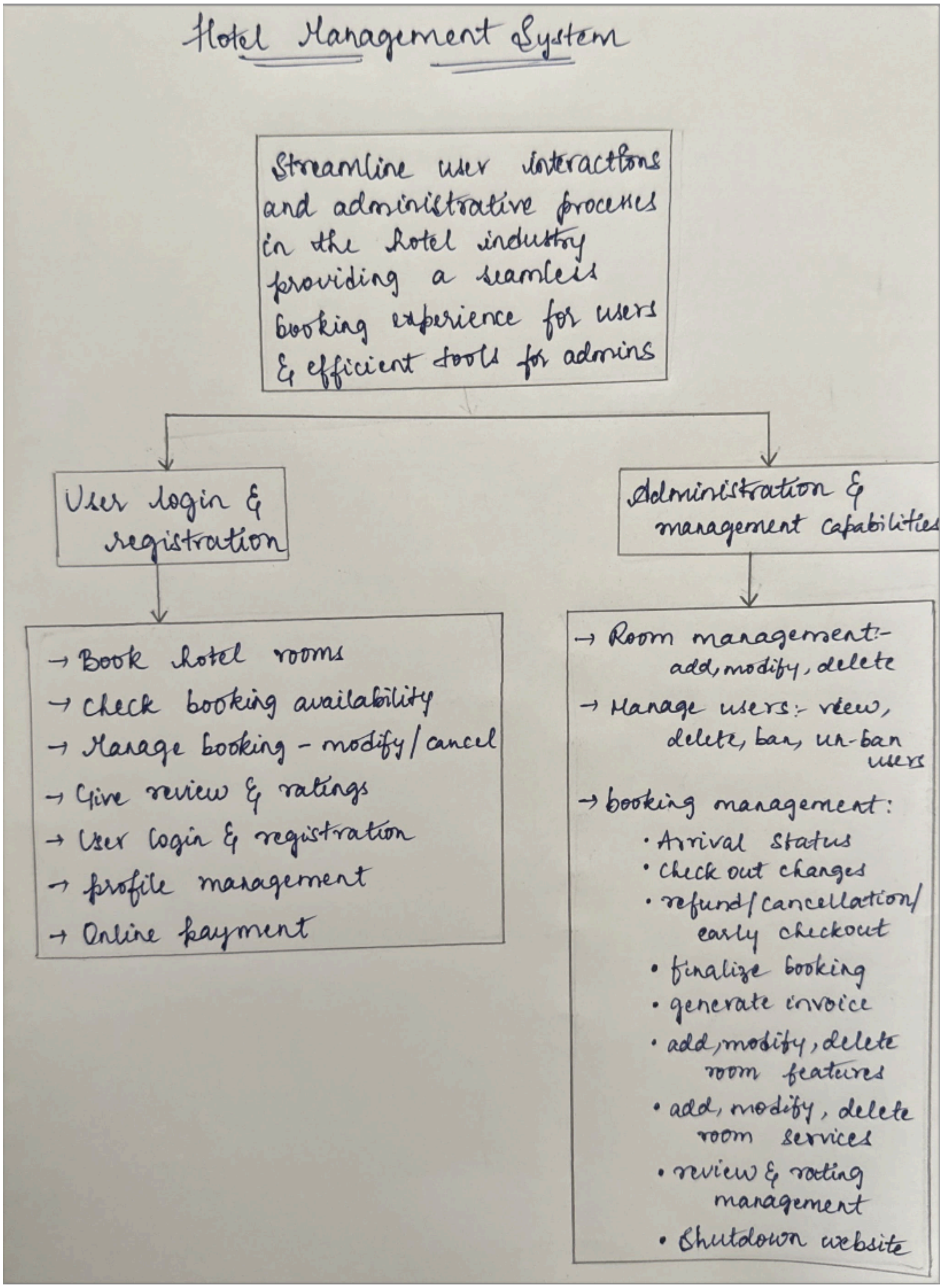
SRM University–AP

Andhra Pradesh – 522 240

May, 2024

Table of Contents

User Story:



Hotel Management System

Streamline user interactions and administrative processes in the hotel industry providing a seamless booking experience for users & efficient tools for admins

User login & registration

- → Book hotel rooms
- → Check booking availability
- → Manage booking - modify/cancel
- → Give review & ratings
- → User login & registration
- → profile management
- → Online payment

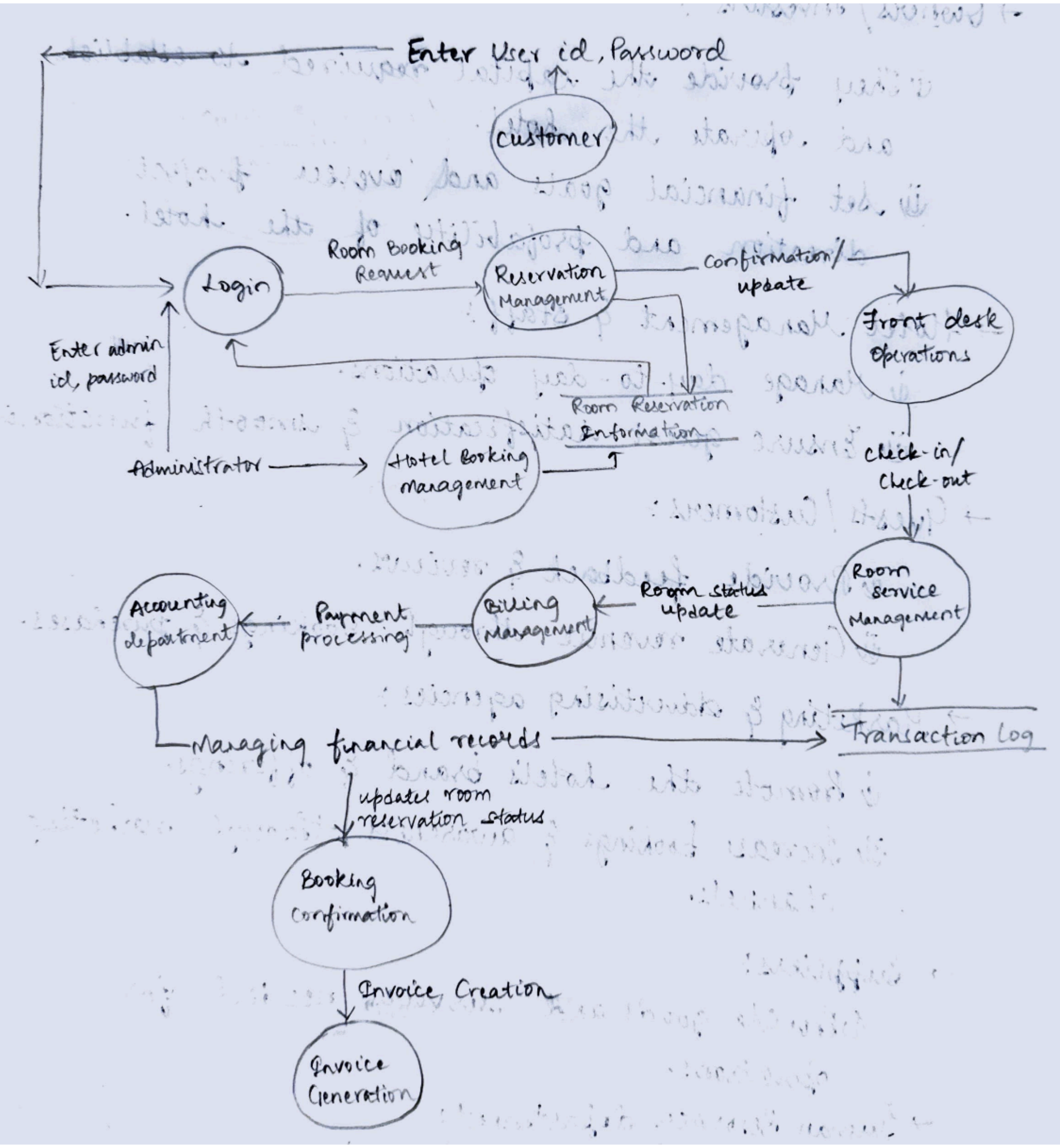
Administration & management capabilities

- → Room management:- add, modify, delete
- → Manage users:- view, delete, ban, un-ban users
- → booking management:
  - Arrival status
  - Check out changes
  - refund/cancellation/early checkout
  - finalize booking
  - generate invoice
  - add, modify, delete room features
  - add, modify, delete room services
  - review & rating management
  - Shutdown website

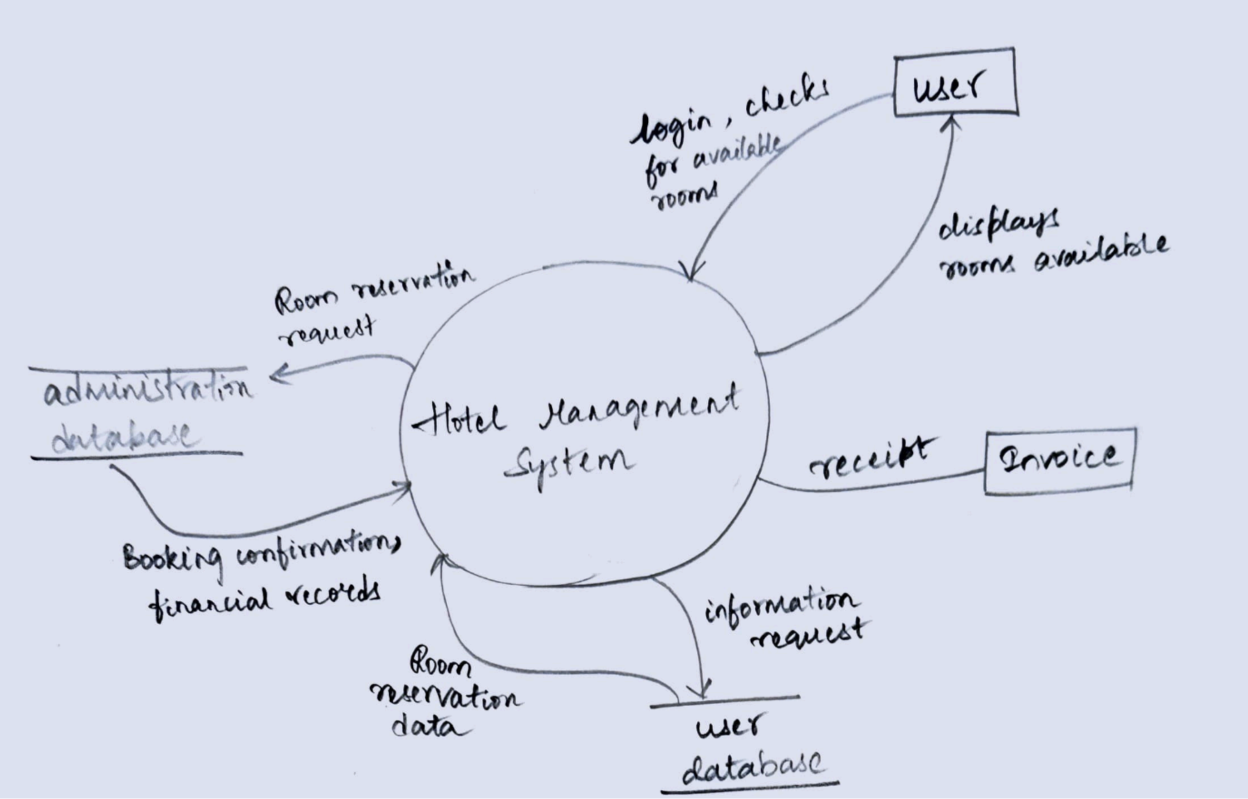
Finding out Stakeholders and their roles in Hotel Management System:

- Owners/ Investors :
    - They provide the capital required to establish and operate the hotel.
    - Set financial goals and oversee project direction and probability of the hotel.
- Hotel Management & Staff:
    - Manage day-to-day operations.
    - Ensure guest satisfaction & smooth functioning.
- Guests/ Customers:
    - Provide feedback & reviews.
    - Generate revenue through bookings & purchases.
- Marketing & Advertising agencies:
    - Promote the hotel's brand & offerings.
    - Increase bookings & awareness through marketing channels.
- Supplies:-
    - Provide goods and services needed for operations.
- Human Resources department:
    - Manages Employee information.
    - Payroll processing.

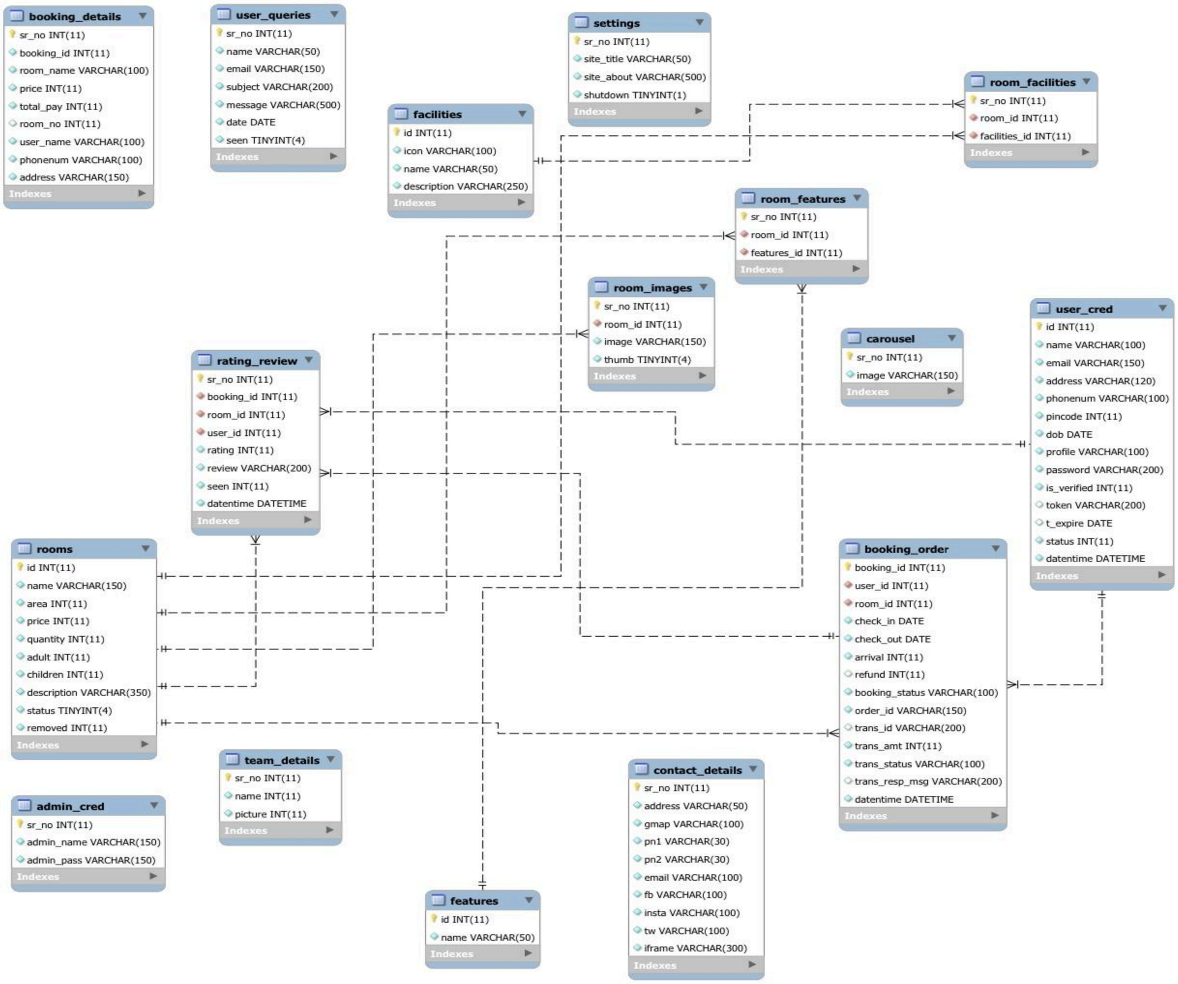
Data Flow diagram:



Context Diagram:



ER_Diagram Entities, and Attributes:



- Uses (Strong Entity)
  → Attributes: Username, Password, Email, firstname, Lastname, phone number, userID.
- Room (strong Entity)
  → Attributes: Room number, type, description price, capacity, availability.
- Facilities (Weak entity of room)
  → Attributes: name, decription, facility ID.
- Booking (Strong entity)
  → Attributes: booking ID, userID(foreignkey) , room number
  (forcin key) , check-in date, check-out date, total price, status.
- Review & Rating (Strong entity)
  → Attributes: reviewID, userID (foreign key) , room number (foreign key),
  rating, review text.
- About us (Strong entity)
  → Attributes: Team members, description
- Contact us (strong entity)
  → Attributes: address, phone number, email, map location,
  social accounts
- Login (Strong entity)
  → Attributes: loginID, user ID (foreign key) ,
- Registration (Strong entity)
  → Attributes: registrationID, registration method, userID (foreign key)

- Payment Gateway (Strong entity)
  → Attributes: paymentID, userID, bookingID (foreign key) ,
  payment method, amount, transactionID, status.
- Invoice (Strong entity)
  → Attributes: inovice ID, booking ID (foreign key) , total
  amount, date, payment status, description, customer name.
- Admin
  → Attributes: adminID, username, password, settings,
  dashboard.

Applying normalization on tables:

Table 1: **admin_cred**

- Step 1: Analyze Functional Dependencies:
    - Primary Key: sr_no
    - Attributes: admin_name, admin_pass
    - Functional Dependencies: { sr_no } -> { admin_name, admin_pass }
- Step 2: Normalize to First Normal Form (1NF):
    - Table already in 1NF as it has a primary key (sr_no) and atomic attribute values.
- Step 3: Normalize to Second Normal Form (2NF):
    - No partial dependencies observed, so the table remains in 2NF.
- Step 4: Normalize to Third Normal Form (3NF):
    - No transitive dependencies found, thus the table remains in 3NF.
- Final Normalized admin_cred Table:
    - admin_cred (sr_no PK, admin_name, admin_pass)

Table 2: **booking_details**

- Step 1: Identify Functional Dependencies:
    - Primary Key: sr_no
    - Attributes: booking_id, room_name, price, total_pay, room_no, user_name, phonenum, address
    - Functional Dependency: { sr_no } -> { booking_id, room_name, price, total_pay, room_no, user_name, phonenum, address }
- Step 2: Normalize to First Normal Form (1NF):
    - No changes needed as the table already has a primary key (sr_no) and atomic attribute values.
- Step 3: Normalize to Second Normal Form (2NF):
    - No partial dependencies observed, so the table remains in 2NF.
- Step 4: Normalize to Third Normal Form (3NF):
    - No transitive dependencies found, thus the table remains in 3NF.
- Final Normalized booking_details Table:
    - booking_details (sr_no PK, booking_id, room_name, price, total_pay, room_no, user_name, phonenum, address)

Table 3: **booking_order**

- Step 1: Analyze Functional Dependencies:
    - Primary Key: booking_id
    - Attributes: user_id, room_id, check_in, check_out, arrival, refund, booking_status, order_id, trans_id, trans_amt, trans_status, trans_resp_msg, datetime
    - Functional Dependencies:

  - { booking_id } -> { user_id, room_id, check_in, check_out, arrival, refund, booking_status, order_id, trans_id, trans_amt, trans_status, trans_resp_msg, datetime }
- Step 2: Normalize to First Normal Form (1NF):
  - No changes needed as the table already has a primary key (booking_id) and atomic attribute values.
- Step 3: Normalize to Second Normal Form (2NF):
  - No partial dependencies observed, so the table remains in 2NF.
- Step 4: Normalize to Third Normal Form (3NF):
  - No transitive dependencies found, thus the table remains in 3NF.
- Step 5: Introduce Foreign Key Constraints:
  - Add foreign key constraints to maintain referential integrity:
    - user_id references user_cred.id
    - room_id references rooms.id
- Final Normalized booking_order Table:
  - booking_order (booking_id PK, user_id FK, room_id FK, check_in, check_out, arrival, refund, booking_status, order_id, trans_id, trans_amt, trans_status, trans_resp_msg, datetime)

Table 4: **carousel**

- Step 1: Analyze Functional Dependencies:
  - Primary Key: sr_no
  - Attributes: image
  - Functional Dependencies: { sr_no } -> { image }
- Step 2: Normalize to First Normal Form (1NF):
  - No changes needed as the table already has a primary key (sr_no) and atomic attribute values.
- Step 3: Normalize to Second Normal Form (2NF):
  - No partial dependencies observed, so the table remains in 2NF.
- Step 4: Normalize to Third Normal Form (3NF):
  - No transitive dependencies found, thus the table remains in 3NF.
- Final Normalized carousel Table:
  - carousel (sr_no PK, image)

Table 5: **contact_details**

- Step 1: Analyze Functional Dependencies:
  - Primary Key: sr_no
  - Attributes: address, gmap, pn1, pn2, email, fb, insta, tw, iframe
  - Functional Dependencies: { sr_no } -> { address, gmap, pn1, pn2, email, fb, insta, tw, iframe }

- Step 2: Normalize to First Normal Form (1NF):
    - No changes needed as the table already has a primary key (sr_no) and atomic attribute values.
- Step 3: Normalize to Second Normal Form (2NF):
    - No partial dependencies observed, so the table remains in 2NF.
- Step 4: Normalize to Third Normal Form (3NF):
    - No transitive dependencies found, thus the table remains in 3NF.
- Final Normalized contact_details Table:
    - contact_details (sr_no PK, address, gmap, pn1, pn2, email, fb, insta, tw, iframe)

Table 6: **facilities**

- Step 1: Analyze Functional Dependencies:
    - Primary Key: id
    - Attributes: icon, name, description
    - Functional Dependencies: { id } -> { icon, name, description }
- Step 2: Normalize to First Normal Form (1NF):
    - No changes needed as the table already has a primary key (id) and atomic attribute values.
- Step 3: Normalize to Second Normal Form (2NF):
    - No partial dependencies observed, so the table remains in 2NF.
- Step 4: Normalize to Third Normal Form (3NF):
    - No transitive dependencies found, thus the table remains in 3NF.
- Final Normalized facilities Table:
    - facilities (id PK, icon, name, description)

Table 7: **features**

- Step 1: Analyze Functional Dependencies:
    - Primary Key: id
    - Attributes: name
    - Functional Dependencies: { id } -> { name }
- Step 2: Normalize to First Normal Form (1NF):
    - No changes needed as the table already has a primary key (id) and atomic attribute values.
- Step 3: Normalize to Second Normal Form (2NF):
    - No partial dependencies observed, so the table remains in 2NF.
- Step 4: Normalize to Third Normal Form (3NF):
    - No transitive dependencies found, thus the table remains in 3NF.
- Final Normalized features Table:
    - features (id PK, name)

Table 8: **rating_review**

- Step 1: Analyze Functional Dependencies:
  - Primary Key: sr_no
  - Attributes: booking_id, room_id, user_id, rating, review, seen, datetime
  - Functional Dependencies:
  - { sr_no } -> { booking_id, room_id, user_id, rating, review, seen, datetime }
- Step 2: Normalize to First Normal Form (1NF):
  - No changes needed as the table already has a primary key (sr_no) and atomic attribute values.
- Step 3: Normalize to Second Normal Form (2NF):
  - No partial dependencies observed, so the table remains in 2NF.
- Step 4: Normalize to Third Normal Form (3NF):
  - No transitive dependencies found, thus the table remains in 3NF.
- Step 5: Introduce Foreign Key Constraints:
  - Add foreign key constraints to maintain referential integrity:
    - booking_id references booking_order.booking_id
    - room_id references rooms.id
    - user_id references user_cred.id
- Final Normalized rating_review Table:
  - rating_review (sr_no PK, booking_id FK, room_id FK, user_id FK, rating, review, seen, datentime)

Table 9: **rooms**

- Step 1: Analyze Functional Dependencies:
- Primary Key: id
- Attributes: name, area, price, quantity, adult, children, description, status, removed
- Functional Dependencies: { id } -> { name, area, price, quantity, adult, children, description, status, removed }
- Step 2: Normalize to First Normal Form (1NF):
  - No changes needed as the table already has a primary key (id) and atomic attribute values.
- Step 3: Normalize to Second Normal Form (2NF):
  - No partial dependencies observed, so the table remains in 2NF.
- Step 4: Normalize to Third Normal Form (3NF):
  - No transitive dependencies found, thus the table remains in 3NF.
- Final Normalized rooms Table:
  - rooms (id PK, name, area, price, quantity, adult, children, description, status, removed)

Table 10: **room_facilities**

- Step 1: Analyze Functional Dependencies:
  - Identify primary key: sr_no.
  - Identify attributes: room_id, facilities_id.
  - Functional Dependencies: { sr_no } -> { room_id, facilities_id }.
- Step 2: Normalize to First Normal Form (1NF):
  - No changes needed; the table already has a primary key (sr_no) and atomic attribute values.
- Step 3: Normalize to Second Normal Form (2NF):
  - No partial dependencies observed, so the table remains in 2NF.
- Step 4: Normalize to Third Normal Form (3NF):
  - No transitive dependencies found, thus the table remains in 3NF.
- Step 5: Add Foreign Key Constraints:
  - Add foreign key constraint to room_id referencing id in the rooms table.
  - Add foreign key constraint to facilities_id referencing id in the facilities table.
- Final Normalized room_facilities Table:
  - room_facilities (sr_no PK, room_id FK, facilities_id FK)

Table 11: **room_features**

- Step 1: Analyze Functional Dependencies:
  - Identify primary key: sr_no.
  - Identify attributes: room_id, features_id.
  - Functional Dependencies: { sr_no } -> { room_id, features_id }.
- Step 2: Normalize to First Normal Form (1NF):
  - No changes needed; the table already has a primary key (sr_no) and atomic attribute values.
- Step 3: Normalize to Second Normal Form (2NF):
  - No partial dependencies observed, so the table remains in 2NF.
- Step 4: Normalize to Third Normal Form (3NF):
  - No transitive dependencies found, thus the table remains in 3NF.
- Step 5: Add Foreign Key Constraints:
  - Add foreign key constraint: room_id references rooms.id.
  - Add foreign key constraint: features_id references features.id.
- Final Normalized room_features Table:
  - room_features (sr_no PK, room_id FK, features_id FK)

Table 12: **room_images**

- Step 1: Analyze Functional Dependencies:
  - Identify primary key: sr_no.
  - Identify attributes: room_id, image, thumb.

- Functional Dependencies: { sr_no } -> { room_id, image, thumb }.

- Step 2: Normalize to First Normal Form (1NF):
  - No changes needed; the table already has a primary key (sr_no) and atomic attribute values.
- Step 3: Normalize to Second Normal Form (2NF):
  - No partial dependencies observed, so the table remains in 2NF.
- Step 4: Normalize to Third Normal Form (3NF):
  - No transitive dependencies found, thus the table remains in 3NF.
- Step 5: Add Foreign Key Constraints:
  - Add foreign key constraint: room_id references rooms.id.
- Final Normalized room_images Table:
  - room_images (sr_no PK, room_id FK, image, thumb)

Table 13: **settings**

- Step 1: Analyze Functional Dependencies:
  - Identify primary key: sr_no.
  - Identify attributes: site_title, site_about, shutdown.
  - Functional Dependencies: { sr_no } -> { site_title, site_about, shutdown }.
- Step 2: Normalize to First Normal Form (1NF):
  - No changes needed; the table already has a primary key (sr_no) and atomic attribute values.
- Step 3: Normalize to Second Normal Form (2NF):
  - No partial dependencies observed, so the table remains in 2NF.
- Step 4: Normalize to Third Normal Form (3NF):
  - No transitive dependencies found, thus the table remains in 3NF.
- Final Normalized settings Table:
  - settings (sr_no PK, site_title, site_about, shutdown)

Table 14: **team_details**

- Step 1: Analyze Functional Dependencies:
  - Identify primary key: sr_no.
  - Identify attributes: name, picture.
  - Functional Dependencies: { sr_no } -> { name, picture }.
- Step 2: Normalize to First Normal Form (1NF):
  - No changes needed; the table already has a primary key (sr_no) and atomic attribute values.
- Step 3: Normalize to Second Normal Form (2NF):
  - No partial dependencies observed, so the table remains in 2NF.
- Step 4: Normalize to Third Normal Form (3NF):
  - No transitive dependencies found, thus the table remains in 3NF.

- Final Normalized team_details Table:
    - team_details (sr_no PK, name, picture)

Table 15: **user_cred**

- Step 1: Analyze Functional Dependencies:
    - Identify primary key: id.
    - Identify attributes: name, email, address, phonenum, pincode, dob, profile, password, is_verified, token, t_expire, status, datentime.
    - Functional Dependencies: { id } -> { name, email, address, phonenum, pincode, dob, profile, password, is_verified, token, t_expire, status, datentime }.
- Step 2: Normalize to First Normal Form (1NF):
    - No changes needed; the table already has a primary key (id) and atomic attribute values.
- Step 3: Normalize to Second Normal Form (2NF):
    - No partial dependencies observed, so the table remains in 2NF.
- Step 4: Normalize to Third Normal Form (3NF):
    - No transitive dependencies found, thus the table remains in 3NF.
- Final Normalized user_cred Table:
    - user_cred (id PK, name, email, address, phonenum, pincode, dob, profile, password, is_verified, token, t_expire, status, datentime)

Table 16: **user_queries**

- Step 1: Analyze Functional Dependencies:
    - Identify primary key: sr_no.
    - Identify attributes: name, email, subject, message, date, seen.
    - Functional Dependencies: { sr_no } -> { name, email, subject, message, date, seen }.
- Step 2: Normalize to First Normal Form (1NF):
    - No changes needed; the table already has a primary key (sr_no) and atomic attribute values.
- Step 3: Normalize to Second Normal Form (2NF):
    - No partial dependencies observed, so the table remains in 2NF.
- Step 4: Normalize to Third Normal Form (3NF):
    - No transitive dependencies found, thus the table remains in 3NF.
- Final Normalized user_queries Table:
    - user_queries (sr_no PK, name, email, subject, message, date, seen)

Architectural Design Analysis for Hotel Management System:

1. **Collect Scenarios:**
    - User End:
      • Scenario 1: A guest registers on the platform, providing personal information for booking.
      • Scenario 2: A guest searches for available rooms, filtering by date, room type, and amenities.
      • Scenario 3: A guest makes a reservation, selecting room preferences and specifying check-in/check-out dates.
      • Scenario 4: User gives reviews and ratings for booked rooms and services.
      • Scenario 5: A guest checks in at the hotel, confirming their reservation and providing identification.
      • Scenario 6: User makes online payments using the Paytm payment gateway.
    - Admin End:
      • Scenario 1: Admin manages room details, including adding, modifying, or deleting room information.
      • Scenario 2: Admin manages users, including viewing, deleting, banning, or un-banning users.
      • Scenario 3: Admin manages bookings, handles arrival status, changes check-out, and processes refunds on cancellations or early check-outs.
      • Scenario 4: Admin finalizes bookings, generates invoices, and manages room features and services.
      • Scenario 5: Admin manages reviews and ratings.
      • Scenario 6: Admin has the ability to shut down the website
2. **Elicit Requirements, Constraints, and Environment Description:**
    - Requirements:
      • Secure user authentication and authorization.
      • Efficient room search and booking functionality.
      • Smooth reservation process with various room options and pricing.
      • Scalability to handle peak booking periods.
      • Usability with a user-friendly interface.
    - Constraints:
      • Limited budget for infrastructure.
      • Use of specific technologies for front end (HTML, CSS, JavaScript, AJAX, Bootstrap 5) and back end (PHP, MySQL).
      Environment:
      • Cloud-based hosting with expected high traffic during peak booking Periods.

3. **Describe Architectural Styles/Patterns:**
    - Module View:
        - Divided into modules such as user management, room management, reservation processing, check-in/out, payment processing, and administrative tasks.
    - Process View:
        - Processes include user registration, room search, booking management, check-in/out procedures, payment processing, review management.
    - Data Flow View:
        - Data flows from the user interface to the server, database queries for room availability, reservation updates, payment processing and check-in/out processes.
4. **Evaluate Quality Attributes:**
    - Security:
      - Ensure secure authentication, encryption of sensitive data, and protection against common web vulnerabilities.
    - Performance:
      - Support efficient room search and booking processes, fast payment processing with minimal latency during peak booking periods.
    - Scalability:
      - Ability to scale horizontally to handle increased booking traffic during peak seasons.
    - Usability:
      - Provide a user-friendly interface with intuitive navigation and responsive design.
5. **Identify Sensitivity of Quality Attributes:**
    - Choice of database affects scalability and performance.
    - Modular architecture allows for scalability but requires careful consideration of data consistency and transaction management.
    - Front-end caching improves performance but may require synchronization with backend updates.
6. **Critique Candidate Architectures:**
    - Candidate 1:
      - Monolithic architecture with relational databases.
      - **Pros:** Simplified development and deployment, lower operational overhead.
      - **Cons:** Limited scalability, potential single point of failure, difficulty in updating individual components.
    - Candidate 2:

- Microservices architecture with NoSQL database.
- **Pros:** Improved scalability, flexibility to update and deploy individual services.
- **Cons:** Increased complexity in development and deployment, higher operational overhead, potential consistency issues with eventual consistency of NoSQL databases.

Based on this analysis, a hybrid approach combining aspects of both architectures might be beneficial, leveraging the simplicity of a monolithic architecture for certain modules while adopting microservices
for scalability-critical components. Additionally, implementing robust security measures and performance optimizations would be essential for ensuring a reliable and efficient hotel management system.

User Interface Designing:

1. Stakeholders:
   → Developers
   → Customers
   → Hotel Owners
   → Payment Gateway providers
   → Investors

2. Requirement Analysis:
   → Home page: provides overview of hotels offering.
   → Room detail page: displaying pricing information & any special offers available for the rooms.
   → User account Management: enable users to create
   accounts or log in with existing accounts & access information.
   → contact and support: link or button in the navigation menu for easy access & provides links to social media platforms and customer support contacts.
   → About us, Facilities: provides information about hotel,
   such as amenities, services & commitment to customer satisfaction. Also displays user feedback & ratings.

- In admin side:
  → Room management
  → Booking management
  → Shutdown festare.

3. Strategies for designing,
   - User- Centered design :
     → Prioritize the needs of both users & administrators when designing.
     → Conduct user research, surveys & usability testing to gather information.
   - Consistent Branding :
     → reflects hotels' brand identity; like colors, topography, logos.
   - Security Measures:
     → Implement robust security to protect users' data.
   - Regular Maintenance & Updates:
     → Establish maintenances schedules for regular
     updates & bug fixes.

4. Quality Attributes:
   → Security
   → Reliability
   → Usability
   → Performance
   → Maintainability
   → Flexibility
   → User Experience

5. Evaluation Techniques :
   → Usability testing
   → Performance testing
   → Code review
   → User acceptance testing
   → User surveys & feedback
   → Continuous monitoring

6. Types of interfaces:
   → GUI
   → Touch screen interface