□ Hotel Management System - Complete Documentation

Project Name: Hotel Management System

Version: 1.0

Date: October 31, 2025

Developer: Shruthana

Technology Stack: React + Node.js + MySQL + Express.js

□ Table of Contents

□ Project Overview

The Hotel Management System is a **full-stack web application** designed to manage all aspects of hotel operations. It provides a comprehensive solution for managing staff, customers, rooms, bookings, payments, services, and analytics.

Key Objectives:

- Streamline hotel operations
- Track bookings and payments efficiently
- Manage staff and customer information
- Provide real-time analytics and insights
- Role-based access control for security
- User-friendly interface with modern design

Target Users:

- Hotel Administrators
- Managers
- Receptionists
- Front desk staff

□ Architecture

Technology Stack

Frontend Technologies:

- **React 18.2.0** - JavaScript library for building user interfaces
- **Vite 4.5.14** - Next-generation frontend build tool
- **React Router v6.20.1** - Declarative routing for React applications
- **Tailwind CSS** - Utility-first CSS framework
- **Lucide React** - Beautiful & consistent icon library
- **Axios** - Promise-based HTTP client

Backend Technologies:

- **Node.js v20.17.0** - JavaScript runtime environment
- **Express.js 4.18.2** - Fast, minimalist web framework
- **MySQL** - Relational database management system
- **mysql2** - MySQL client for Node.js
- **CORS** - Cross-Origin Resource Sharing middleware
- **body-parser** - Request body parsing middleware

Development Tools:

- **npm** - Package manager
- **Git** - Version control
- **VS Code** - Code editor
- **MySQL Workbench** - Database management

Application Architecture

```
┌─────────────────────────────────────────────┐
│           Browser (Client)                   │
│                                              │
│  ┌─────────────────────────────────────┐    │
│  │   React Application (Frontend)       │    │
│  │   - Components                       │    │
│  │   - Pages                            │    │
│  │   - Context (State Management)       │    │
│  │   - API Service Layer                │    │
│  └─────────────────────────────────────┘    │
└─────────────────────────────────────────────┘
                   ↓ HTTP/HTTPS
┌─────────────────────────────────────────────┐
│        Node.js Server (Backend)              │
│                                              │
│  ┌─────────────────────────────────────┐    │
│  │   Express.js Application             │    │
│  │   - REST API Routes                  │    │
│  │   - Authentication Middleware        │    │
│  │   - Business Logic                   │    │
│  │   - Static File Serving              │    │
│  └─────────────────────────────────────┘    │
└─────────────────────────────────────────────┘
                   ↓ SQL Queries
┌─────────────────────────────────────────────┐
│           MySQL Database                     │
│                                              │
│  - staff                                     │
│  - customers                                 │
│  - rooms                                     │
│  - bookings                                  │
│  - payments                                  │
│  - services                                  │
│  - booking_services                          │
└─────────────────────────────────────────────┘
```

Deployment Architecture

- **Single Port Deployment (3001)**
  - Backend serves both API endpoints and frontend static files
  - React Router integrated with Express catch-all route
  - Production-ready configuration

□ Project Structure

```
hotel_management_system/
│
├── hotel-management-backend/          # Backend API server
│   ├── server.js                      # Main Express server file
│   ├── package.json                   # Backend dependencies
│   │
│   ├── database/                      # Database files
│   │   ├── connection.js             # MySQL database connection
│   │   ├── setup.sql                 # Database schema & initial data
│   │   └── update_names.sql          # Karnataka names update script
│   │
│   ├── routes/                        # API route handlers
│   │   ├── auth.js                   # Authentication routes
│   │   ├── staff.js                  # Staff CRUD operations
│   │   ├── customers.js              # Customer management
│   │   ├── rooms.js                  # Room management
│   │   ├── bookings.js               # Booking operations
│   │   ├── payments.js               # Payment processing
│   │   ├── services.js               # Hotel services management
│   │   ├── bookingServices.js        # Booking-service linking
│   │   └── analytics.js              # Analytics & reporting
│   │
│   └── utils/                         # Utility functions
│       └── helpers.js                # Helper functions
│
└── hotel-management-frontend/         # React frontend application
    ├── package.json                   # Frontend dependencies
    ├── vite.config.js                # Vite configuration
    ├── tailwind.config.js            # Tailwind CSS configuration
    ├── index.html                    # HTML template
    │
    ├── src/                          # Source code
    │   ├── App.jsx                   # Main app component with routing
    │   ├── main.jsx                  # React entry point
    │   ├── index.css                 # Global styles
    │   │
    │   ├── components/               # Reusable components
    │   │   ├── common/              # Common UI components
    │   │   │   ├── Header.jsx       # Top navigation bar
    │   │   │   ├── Sidebar.jsx      # Left sidebar menu
    │   │   │   ├── Layout.jsx       # Page layout wrapper
    │   │   │   ├── Modal.jsx        # Modal dialog component
    │   │   │   ├── LoadingSpinner.jsx # Loading indicator
    │   │   │   └── ProtectedRoute.jsx # Authentication guard
    │   │   │
    │   │   ├── forms/               # Form components
    │   │   │   ├── LoginForm.jsx    # Login form
    │   │   │   ├── StaffForm.jsx    # Staff form
    │   │   │   ├── CustomerForm.jsx # Customer form
    │   │   │   ├── RoomForm.jsx     # Room form
    │   │   │   ├── BookingForm.jsx  # Booking form
    │   │   │   └── PaymentForm.jsx  # Payment form
    │   │   │
    │   │   └── tables/              # Table components
    │   │       ├── StaffTable.jsx   # Staff data table
│   │       ├── CustomersTable.jsx # Customer data table
│   │       ├── RoomsTable.jsx   # Room data table
│   │       ├── BookingsTable.jsx # Booking data table
│   │       └── PaymentsTable.jsx # Payment data table
│   │
│   ├── context/                 # React Context for state
│   │   └── AuthContext.jsx     # Authentication context
│   │
│   ├── pages/                   # Page components
│   │   ├── Dashboard.jsx       # Main dashboard
│   │   │
│   │   ├── Auth/               # Authentication pages
│   │   │   └── Login.jsx       # Login page
│   │   │
│   │   ├── Staff/              # Staff management pages
│   │   │   ├── StaffList.jsx   # Staff listing
│   │   │   └── StaffFormPage.jsx # Add/Edit staff
│   │   │
│   │   ├── Customers/          # Customer management pages
│   │   │   ├── CustomersList.jsx # Customer listing
│   │   │   └── CustomerFormPage.jsx # Add/Edit customer
│   │   │
│   │   ├── Rooms/              # Room management pages
│   │   │   ├── RoomsList.jsx   # Room listing
│   │   │   └── RoomFormPage.jsx # Add/Edit room
│   │   │
│   │   ├── Bookings/           # Booking management pages
│   │   │   ├── BookingsList.jsx # Booking listing
│   │   │   └── BookingFormPage.jsx # Create booking
│   │   │
│   │   ├── Payments/           # Payment management pages
│   │   │   └── PaymentsList.jsx # Payment listing
│   │   │
│   │   ├── Services/           # Services management pages
│   │   │   └── ServicesList.jsx # Services listing
│   │   │
│   │   └── Analytics/          # Analytics pages
│   │       └── AnalyticsDashboard.jsx # Analytics dashboard
│   │
│   ├── services/               # API service layer
│   │   └── api.js             # API client functions
│   │
│   └── utils/                  # Utility functions
│       └── helpers.js         # Helper functions
│
└── dist/                       # Production build output
    ├── index.html
    └── assets/                # Compiled CSS & JS
```

□ Database Schema

Database Name: `hotel_management_system`

Tables Overview:

1. **staff** - Hotel Employees

Stores information about hotel staff members with different roles and access levels.

Fields:

- `staff_id` (INT, Primary Key, AUTO_INCREMENT) - Unique staff identifier
- `username` (VARCHAR(50), UNIQUE) - Login username
- `password` (VARCHAR(255)) - Login password
- `full_name` (VARCHAR(100)) - Employee full name
- `email` (VARCHAR(100), UNIQUE) - Email address
- `phone` (VARCHAR(15)) - Contact number
- `role` (ENUM) - Staff role (Super Admin, Admin, Manager, Receptionist)
- `salary` (DECIMAL(10,2)) - Monthly salary
- `created_at` (TIMESTAMP) - Record creation timestamp

Sample Data:

- Rajesh Kumar (Admin)
- Kavya Reddy (Manager)
- Priya Shetty (Receptionist)

---

2. **customers** - Hotel Guests

Maintains customer/guest information with identification details.

Fields:

- `customer_id` (INT, Primary Key, AUTO_INCREMENT) - Unique customer identifier
- `first_name` (VARCHAR(50)) - Customer first name
- `last_name` (VARCHAR(50)) - Customer last name
- `email` (VARCHAR(100), UNIQUE) - Email address
- `phone` (VARCHAR(15)) - Contact number
- `address` (TEXT) - Residential address
- `id_proof_type` (VARCHAR(50)) - Type of ID (Aadhar, Passport, Driver License, PAN Card)
- `id_proof_number` (VARCHAR(50)) - ID number
- `date_of_birth` (DATE) - Date of birth
- `created_at` (TIMESTAMP) - Record creation timestamp

Sample Data:

- Vikram Shetty
- Meera Nayak
- Kiran Kumar

---

3. **rooms** - Hotel Room Inventory

Manages hotel rooms with different types and pricing.

Fields:

- `room_id` (INT, Primary Key, AUTO_INCREMENT) - Unique room identifier
- `room_number` (VARCHAR(10), UNIQUE) - Room number
- `room_type` (ENUM) - Room category (Standard, Deluxe, Suite, Executive, Presidential Suite)
- `price_per_night` (DECIMAL(10,2)) - Nightly rate in ₹

- `status` (ENUM) - Room status (Available, Occupied, Maintenance)
- `description` (TEXT) - Room description and amenities
- `created_at` (TIMESTAMP) - Record creation timestamp

Room Types & Pricing:

- Standard: ₹2,500 - ₹3,000
- Deluxe: ₹4,000 - ₹5,000
- Suite: ₹6,000 - ₹8,000
- Executive: ₹8,000 - ₹10,000
- Presidential Suite: ₹15,000+

---

4. **bookings** - Room Reservations

Tracks all room bookings and reservations.

Fields:

- `booking_id` (INT, Primary Key, AUTO_INCREMENT) - Unique booking identifier
- `customer_id` (INT, Foreign Key → customers) - Customer reference
- `room_id` (INT, Foreign Key → rooms) - Room reference
- `check_in` (DATE) - Check-in date
- `check_out` (DATE) - Check-out date
- `total_nights` (INT) - Number of nights
- `total_amount` (DECIMAL(10,2)) - Total booking amount in ₹
- `status` (ENUM) - Booking status (Confirmed, Checked-in, Checked-out, Cancelled)
- `special_requests` (TEXT) - Special requirements
- `created_at` (TIMESTAMP) - Record creation timestamp

Status Flow:

1. Confirmed - Booking created
2. Checked-in - Guest has checked in
3. Checked-out - Guest has checked out
4. Cancelled - Booking cancelled

---

5. **payments** - Payment Transactions

Records all payment transactions for bookings.

Fields:

- `payment_id` (INT, Primary Key, AUTO_INCREMENT) - Unique payment identifier
- `booking_id` (INT, Foreign Key → bookings) - Booking reference
- `amount` (DECIMAL(10,2)) - Payment amount in ₹
- `payment_date` (DATETIME) - Payment date and time
- `payment_method` (ENUM) - Payment method (Cash, Credit Card, Debit Card, UPI, Net Banking)
- `payment_status` (ENUM) - Payment status (Pending, Completed, Failed, Refunded)
- `transaction_id` (VARCHAR(100)) - Transaction reference
- `created_at` (TIMESTAMP) - Record creation timestamp

Payment Methods:

- Cash
- Credit Card
- Debit Card
- UPI
- Net Banking

6. **services** - Hotel Services Catalog

Maintains catalog of additional hotel services.

Fields:

- `service_id` (INT, Primary Key, AUTO_INCREMENT) - Unique service identifier
- `service_name` (VARCHAR(100)) - Service name
- `description` (TEXT) - Service description
- `category` (VARCHAR(50)) - Service category (Food, Spa, Laundry, Transportation, Others)
- `price` (DECIMAL(10,2)) - Service price in ₹
- `created_at` (TIMESTAMP) - Record creation timestamp

Service Categories:

- Food & Beverage
- Spa & Wellness
- Laundry
- Transportation
- Room Service
- Event Services

---

7. **booking_services** - Service Bookings

Links services to bookings for additional charges.

Fields:

- `booking_service_id` (INT, Primary Key, AUTO_INCREMENT) - Unique identifier
- `booking_id` (INT, Foreign Key → bookings) - Booking reference
- `service_id` (INT, Foreign Key → services) - Service reference
- `quantity` (INT) - Service quantity
- `total_price` (DECIMAL(10,2)) - Total service price in ₹
- `created_at` (TIMESTAMP) - Record creation timestamp

Database Relationships:

```
customers (1) ──────── (N) bookings
rooms (1) ──────── (N) bookings
bookings (1) ──────── (N) payments
bookings (1) ──────── (N) booking_services
services (1) ──────── (N) booking_services
```

□ Authentication & Authorization

Authentication System

The application uses a **session-based authentication** system with localStorage.

Login Flow:

1. User enters `username` and `password` on login page
2. Frontend sends POST request to `/api/auth/login`

3. Backend validates credentials against `staff` table
4. On success, returns user object with role and permissions
5. Frontend stores authentication data in localStorage:
    - `token` - Authentication token
    - `staffId` - Staff ID
    - `user` - User object (full_name, role, email, etc.)
6. User is redirected to dashboard

Logout Flow:

1. User clicks Logout button in sidebar
2. Frontend clears localStorage (token, staffId, user)
3. User state set to null
4. ProtectedRoute detects null user
5. Automatically redirects to login page

Role-Based Access Control (RBAC)

The system implements four user roles with different permission levels:

1. **Super Admin** (Highest Privileges)

Permissions:

- □ Manage Staff
- □ Manage Customers
- □ Manage Rooms
- □ Manage Bookings
- □ Manage Payments
- □ View Analytics
- □ Manage Services
- □ Delete Records
- □ Edit All Records

Access:

- Full system access
- Can perform all CRUD operations
- Can view all reports and analytics

---

2. **Admin** (High Privileges)

Permissions:

- □ Manage Staff
- □ Manage Customers
- □ Manage Rooms
- □ Manage Bookings
- □ Manage Payments
- □ View Analytics
- □ Manage Services
- □ Delete Records
- □ Edit All Records

Access:

- Same as Super Admin
- Designed for hotel administrators

Login Credentials:

- Username: `admin`
- Password: `admin123`

---

3. **Manager** (Medium Privileges)

Permissions:

- □ Manage Staff (Cannot add/edit/delete staff)
- □ Manage Customers
- □ Manage Rooms
- □ Manage Bookings
- □ Manage Payments
- □ View Analytics
- □ Manage Services
- □ Delete Records (Cannot delete any records)
- □ Edit All Records

Access:

- Cannot manage staff members
- Cannot delete records (safety measure)
- Can view and edit most data
- Full access to analytics

Login Credentials:

- Username: `manager`
- Password: `manager123`

---

4. **Receptionist** (Limited Privileges)

Permissions:

- □ Manage Staff
- □ Manage Customers
- □ Manage Rooms (Cannot add/edit rooms)
- □ Manage Bookings
- □ Manage Payments
- □ View Analytics (Cannot access analytics)
- □ Manage Services
- □ Delete Records
- □ Edit All Records (Limited edit access)

Access:

- Primary function: Front desk operations
- Can manage bookings and payments
- Can register new customers
- Cannot access analytics or staff management
- Cannot modify room inventory

Login Credentials:

- Username: `receptionist`
- Password: `recept123`

Permission Implementation

Permissions are enforced at multiple levels:

1. **Frontend Level:**

- Sidebar menu items hidden based on permissions
- Buttons and actions disabled for unauthorized users
- Protected routes redirect unauthorized access

2. **Backend Level:**

- API endpoints validate user permissions
- Database queries filtered by user role
- Error responses for unauthorized actions

□ Key Features

1. **Dashboard** □

The main landing page after login provides an overview of hotel operations.

Features:

- **Real-time Statistics Cards:**
  - Total Customers Count
  - Total Rooms Count
  - Active Bookings Count
  - Total Revenue (in ₹)
  - Occupancy Rate (%)
  - Monthly Revenue (Last 30 days in ₹)
- **Quick Access Cards:**
  - Navigate to Staff Management
  - Navigate to Customer Management
  - Navigate to Room Management
  - Navigate to Booking Management
  - Navigate to Payment Tracking
  - Navigate to Analytics
- **Role-based Visibility:**
  - Cards shown/hidden based on user permissions
  - Different statistics for different roles

---

2. **Staff Management** □

Comprehensive staff management system for hotel employees.

Features:

- **View All Staff:**
  - Searchable and filterable staff list
  - Display: Name, Email, Phone, Role, Salary
  - Action buttons: Edit, Delete

- **Add New Staff:**
  - Form fields:
    - Username (unique)
    - Password
    - Full Name
    - Email (unique)
    - Phone Number
    - Role (dropdown)
    - Salary (in ₹)
  - Validation for all fields
  - Duplicate username/email prevention
- **Edit Staff:**
  - Update all staff details
  - Optional password update
  - Cannot change username
- **Delete Staff:**
  - Confirmation before deletion
  - Prevents accidental deletion

Access: Admin and Super Admin only

---

3. **Customer Management** □

Maintain comprehensive customer database with identification details.

Features:

- **View All Customers:**
  - Search and filter functionality
  - Display: Name, Email, Phone, ID Proof
  - Karnataka-based customer names
  - Indian phone numbers (10 digits)
- **Add New Customer:**
  - Form fields:
    - First Name
    - Last Name
    - Email (unique)
    - Phone (Indian format)
    - Address
    - ID Proof Type (Aadhar, Passport, Driver License, PAN Card)
    - ID Proof Number
    - Date of Birth
  - Input validation
  - Duplicate email prevention
- **Edit Customer:**
  - Update customer information
  - Maintain booking history
- **Delete Customer:**

- Soft delete to preserve booking history
- Confirmation required

Access: All roles except Receptionist have full access

---

4. **Room Management** □

Manage hotel room inventory, pricing, and availability.

Features:

- **View All Rooms:**
  - Filter by room type
  - Filter by status (Available/Occupied/Maintenance)
  - Display: Room Number, Type, Price per Night, Status
  - Pricing displayed in ₹
- **Add New Room:**
  - Form fields:
    - Room Number (unique)
    - Room Type (Standard, Deluxe, Suite, Executive, Presidential Suite)
    - Price per Night (in ₹)
    - Status (Available, Occupied, Maintenance)
    - Description & Amenities
  - Automatic room number validation
- **Edit Room:**
  - Update room details and pricing
  - Change room status
  - Cannot change room number
- **Delete Room:**
  - Only if no active bookings
  - Confirmation required
- **Room Status Management:**
  - Available - Ready for booking
  - Occupied - Currently booked
  - Maintenance - Under maintenance

Access: Admin, Manager (Receptionist view-only)

---

5. **Booking Management** □

Complete booking and reservation system.

Features:

- **View All Bookings:**
  - Filter by status, date range
  - Display: Customer Name, Room Number, Dates, Amount, Status
  - Color-coded status badges
  - Amount displayed in ₹

- **Create New Booking:**
    - Form fields:
        - Select Customer (dropdown)
        - Select Room (available rooms only)
        - Check-in Date
        - Check-out Date
        - Number of Guests
        - Special Requests
    - Automatic calculations:
        - Total Nights = Check-out - Check-in
        - Total Amount = Nights × Room Price per Night
    - Room availability validation
    - Date validation (check-out > check-in)
- **Edit Booking:**
    - Modify dates and room
    - Update special requests
    - Recalculate amounts
- **Cancel Booking:**
    - Change status to Cancelled
    - Free up room availability
    - Maintain booking history
- **Booking Status:**
    - Confirmed - New booking
    - Checked-in - Guest has arrived
    - Checked-out - Guest has departed
    - Cancelled - Booking cancelled
- **Add Services to Booking:**
    - Link hotel services
    - Add quantity
    - Calculate service charges

Access: All roles (Receptionist primary user)

---

6. **Payment Management** □

Track all payment transactions and revenue.

Features:

- **View All Payments:**
    - Filter by date, status, method
    - Display: Booking ID, Customer, Amount, Date, Method, Status
    - Amounts in ₹
    - Payment status badges
- **Record Payment:**
    - Form fields:
        - Select Booking
        - Payment Amount (in ₹)

        - Payment Date & Time
        - Payment Method (Cash, Credit Card, Debit Card, UPI, Net Banking)
        - Transaction ID
    - Partial payments supported
    - Payment status tracking

- **Payment Status:**
    - Pending - Payment not received
    - Completed - Payment successful
    - Failed - Payment failed
    - Refunded - Amount refunded

- **Payment Methods:**
    - Cash
    - Credit Card
    - Debit Card
    - UPI
    - Net Banking

Access: All roles (essential for operations)

---

7. **Services Management** □

Manage additional hotel services and amenities.

Features:

- **View All Services:**
    - Categorized service listing
    - Display: Service Name, Category, Price, Description
    - Prices in ₹

- **Add New Service:**
    - Form fields:
        - Service Name
        - Description
        - Category (Food, Spa, Laundry, Transportation, Others)
        - Price (in ₹)

- **Edit Service:**
    - Update service details and pricing

- **Delete Service:**
    - Remove unused services

- **Service Categories:**
    - Food & Beverage
    - Spa & Wellness
    - Laundry
    - Transportation
    - Room Service
    - Event Services

Access: Admin, Manager, Receptionist

8. **Analytics Dashboard** □

Real-time analytics and reporting for business insights.

Features:

Key Metrics (Cards):

1. Total Customers - Registered guests count
2. Total Rooms - Hotel capacity
3. Active Bookings - Current confirmed/checked-in bookings
4. Total Revenue - All-time revenue in ₹
5. Occupancy Rate - Percentage of occupied rooms
6. Monthly Revenue - Last 30 days revenue in ₹

Revenue by Room Type (Table):

- Room Type
- Total Bookings
- Total Revenue (₹)
- Helps identify most profitable room categories

Popular Services (Table):

- Service Name
- Category
- Total Bookings
- Total Revenue (₹)
- Identifies most-used services

Premium Customers (Cards):

- Customers who booked expensive rooms
- Display: Name, Email
- Helps identify VIP guests

Recent Bookings (Table):

- Latest 10 bookings
- Customer details
- Room information
- Booking dates and nights
- Total amount (₹)
- Status with color coding

Data Updates:

- Real-time calculations from database
- No cached data
- Automatic refresh on page load

Access: Admin, Manager (Analytics hidden from Receptionist)

□ Data Flow

Example: Creating a New Booking

This section demonstrates how data flows through the system when creating a booking.

Step 1: User Interaction (Frontend)

```
User navigates to Bookings → Create Booking
Fills in the form:
- Customer: Selects from dropdown (e.g., "Vikram Shetty")
- Room: Selects available room (e.g., "Room 101 - Deluxe")
- Check-in: 2025-11-01
- Check-out: 2025-11-03
- Guests: 2
- Special Requests: "Late checkout if possible"
```

Step 2: Frontend Validation

```
- Check all required fields filled
- Validate check-out > check-in
- Calculate total nights: 2 nights
- Fetch room price: ₹4,500/night
- Calculate total amount: ₹9,000
- Display amount to user for confirmation
```

Step 3: API Request

```
POST http://localhost:3001/api/bookings
Headers:
  Content-Type: application/json

Body:
{
  "customer_id": 5,
  "room_id": 3,
  "check_in": "2025-11-01",
  "check_out": "2025-11-03",
  "number_of_guests": 2,
  "special_requests": "Late checkout if possible"
}
```

Step 4: Backend Processing (routes/bookings.js)

```
1. Receive request
2. Extract data from request body
3. Validate data:
   - Customer exists
   - Room exists and is available
   - Check-in date not in past
   - Check-out after check-in
4. Calculate total nights and amount:
   - Total nights = 2
   - Room price = ₹4,500
   - Total amount = 2 × ₹4,500 = ₹9,000
5. Insert into database:
   INSERT INTO bookings (customer_id, room_id, check_in,
   check_out, total_nights, total_amount, status, special_requests)
   VALUES (5, 3, '2025-11-01', '2025-11-03', 2, 9000,
   'Confirmed', 'Late checkout if possible')
6. Update room status to 'Occupied'
7. Return success response with booking_id
```

Step 5: Database Transaction

```sql
START TRANSACTION;

-- Insert booking
INSERT INTO bookings
VALUES (NULL, 5, 3, '2025-11-01', '2025-11-03', 2, 9000.00,
'Confirmed', 'Late checkout if possible', NOW());

-- Update room status
UPDATE rooms
SET status = 'Occupied'
WHERE room_id = 3;

COMMIT;
```

Step 6: Backend Response

```json
{
  "success": true,
  "message": "Booking created successfully",
  "booking": {
    "booking_id": 25,
    "customer_id": 5,
    "room_id": 3,
    "check_in": "2025-11-01",
    "check_out": "2025-11-03",
    "total_nights": 2,
    "total_amount": 9000,
    "status": "Confirmed"
  }
}
```

Step 7: Frontend Response Handling

```
- Display success message: "Booking created successfully!"
- Refresh bookings list
- Navigate back to bookings page
- Show new booking in the table
- Update dashboard statistics
```

API Endpoints

Base URL: `http://localhost:3001/api`

Authentication Endpoints

POST `/auth/login`

Login with username and password

Request Body:

```
{
  "username": "admin",
  "password": "admin123"
}
```

Response:

```
{
  "message": "Login successful",
  "user": {
    "staff_id": 1,
    "username": "admin",
    "full_name": "Rajesh Kumar",
    "email": "rajesh@hotel.com",
    "role": "Admin",
    "phone": "9876543210"
  }
}
```

Staff Endpoints

GET `/staff`

Get all staff members

Response:

```
[
  {
    "staff_id": 1,
    "username": "admin",
    "full_name": "Rajesh Kumar",
    "email": "rajesh@hotel.com",
    "phone": "9876543210",
    "role": "Admin",
    "salary": "50000.00"
  }
]
```

POST `/staff`

Create new staff member

Request Body:

```
{
  "username": "newstaff",
  "password": "password123",
  "full_name": "Arvind Rao",
  "email": "arvind@hotel.com",
  "phone": "9898989898",
  "role": "Receptionist",
  "salary": "25000"
}
```

PUT `/staff/:id`

Update staff member

DELETE `/staff/:id`

Delete staff member

Customer Endpoints

GET `/customers`

Get all customers

Response:

```
[
  {
    "customer_id": 1,
    "first_name": "Vikram",
    "last_name": "Shetty",
    "email": "vikram@gmail.com",
    "phone": "9812345678",
    "address": "Bangalore, Karnataka",
    "id_proof_type": "Aadhar Card",
    "id_proof_number": "1234-5678-9012"
  }
]
```

POST `/customers`

Create new customer

PUT `/customers/:id`

Update customer

DELETE `/customers/:id`

Delete customer

Room Endpoints

GET `/rooms`

Get all rooms

Response:

```
[
  {
    "room_id": 1,
    "room_number": "101",
    "room_type": "Deluxe",
    "price_per_night": "4500.00",
    "status": "Available",
    "description": "Spacious room with city view"
  }
]
```

POST `/rooms`

Create new room

PUT `/rooms/:id`

Update room

DELETE `/rooms/:id`

Delete room

Booking Endpoints

GET `/bookings`

Get all bookings with customer and room details

Response:

```
[
  {
    "booking_id": 1,
    "customer_id": 5,
    "customer_name": "Vikram Shetty",
    "room_id": 3,
    "room_number": "101",
    "check_in": "2025-11-01",
    "check_out": "2025-11-03",
    "total_nights": 2,
    "total_amount": "9000.00",
    "status": "Confirmed"
  }
]
```

POST `/bookings`

Create new booking

Request Body:

```
{
  "customer_id": 5,
  "room_id": 3,
  "check_in": "2025-11-01",
  "check_out": "2025-11-03",
  "number_of_guests": 2,
  "special_requests": "Late checkout"
}
```

PUT `/bookings/:id`

Update booking

DELETE `/bookings/:id`

Cancel booking

Payment Endpoints

GET `/payments`

Get all payments

Response:

```
[
  {
    "payment_id": 1,
    "booking_id": 1,
    "customer_name": "Vikram Shetty",
    "amount": "9000.00",
    "payment_date": "2025-11-01 14:30:00",
    "payment_method": "UPI",
    "payment_status": "Completed",
    "transaction_id": "UPI12345"
  }
]
```

POST `/payments`

Create new payment

Request Body:

```
{
  "booking_id": 1,
  "amount": 9000,
  "payment_method": "UPI",
  "transaction_id": "UPI12345"
}
```

Service Endpoints

GET `/services`

Get all services

POST `/services`

Create new service

PUT `/services/:id`

Update service

DELETE `/services/:id`

Delete service

Analytics Endpoints

GET `/analytics/revenue-by-room-type`

Get revenue breakdown by room type

Response:

```
[
  {
    "room_type": "Deluxe",
    "total_bookings": 15,
    "total_revenue": "67500.00"
  },
  {
    "room_type": "Suite",
    "total_bookings": 8,
    "total_revenue": "48000.00"
  }
]
```

GET `/analytics/popular-services`

Get most booked services

GET `/analytics/premium-customers`

Get customers who booked expensive rooms

GET `/analytics/detailed-bookings`

Get detailed booking information

□ Special Customizations

1. **Currency - Indian Rupee (₹)**

All prices and amounts throughout the system are displayed in Indian Rupees.

Implementation:

- Frontend: All currency displays use ₹ symbol
- Database: DECIMAL(10,2) for precise currency storage
- Formatting: ₹`${amount.toLocaleString()}`

Examples:

- Room Price: ₹4,500/night
- Total Revenue: ₹1,25,000
- Service Price: ₹500

---

2. **Karnataka-Based Names**

All staff and customer names are Karnataka-based for localization.

Staff Names:

- Rajesh Kumar (Admin)
- Kavya Reddy (Manager)
- Priya Shetty (Receptionist)
- Arjun Rao (Housekeeping)
- Lakshmi Hegde (Front Desk)

Customer Names:

- Vikram Shetty
- Meera Nayak
- Kiran Kumar
- Sowmya Rao
- Anil Bhat

Implementation:

- Database update script: `update_to_karnataka_names.js`
- SQL script: `update_names.sql`
- Common surnames: Shetty, Reddy, Rao, Kumar, Hegde, Nayak, Bhat

---

3. **Indian Phone Number Format**

Phone numbers follow Indian mobile number format.

Format:

- 10 digits
- Starts with 9 or 8
- Example: 9876543210, 9812345678

Validation:

- Length: Exactly 10 digits
- Starting digit: 9 or 8
- No special characters or spaces

---

4. **UI Customizations**

Removed Elements:

- Top-right user section (notification bell, profile, admin badge)
- Top-right logout button
- Settings option from sidebar

Added Elements:

- Logout button in sidebar (bottom)
- Red hover effect on logout button
- Clean, minimal header

Design Philosophy:

- Minimal and clean interface
- Focus on functionality
- Easy navigation
- Role-based UI elements

---

5. **Analytics Display - Tables Only**

Analytics page displays all data in table format (no graphs/charts).

Reason:

- User preference for tabular data
- Easier to read exact numbers
- Better for detailed analysis

- No dependency on chart libraries

Components:

- Revenue by Room Type - Table
- Popular Services - Table
- Premium Customers - Grid Cards
- Recent Bookings - Detailed Table

□ How to Run the Project

Prerequisites:

- Node.js v20.17.0 or higher
- MySQL Server running
- npm package manager

Step 1: Database Setup

1. **Start MySQL Server**
2. **Create Database:**

```
CREATE DATABASE hotel_management_system;
```

3. **Import Schema:** Run the SQL file: `hotel-management-backend/database/setup.sql`

This will create all tables and insert sample data.

Step 2: Backend Setup

1. **Navigate to backend folder:**

```
cd "C:\Users\Shruthana\Downloads\hotel_management_system (2)\hotel_management_system\hotel-management-backend"
```

2. **Install dependencies:**

```
npm install
```

3. **Configure database connection:** Edit `database/connection.js` with your MySQL credentials:

```
const connection = mysql.createConnection({
  host: 'localhost',
  user: 'root',
  password: 'your_password',
  database: 'hotel_management_system'
});
```

Step 3: Frontend Setup

1. **Navigate to frontend folder:**

```
cd "C:\Users\Shruthana\Downloads\hotel_management_system (2)\hotel_management_system\hotel-management-frontend"
```

2. **Install dependencies:**

```
npm install
```

3. **Build frontend:**

```
npm run build
```

This creates production-ready files in `dist/` folder.

Step 4: Start the Server

1. **Navigate to backend folder:**

```
cd "C:\Users\Shruthana\Downloads\hotel_management_system (2)\hotel_management_system\hotel-management-backend"
```

2. **Start server:**

```
npm start
```

3. **Server will start on port 3001:**

```
□ Server running on port 3001
□ Health check: http://localhost:3001/health
□ API Base: http://localhost:3001/api
```

Step 5: Access the Application

URL: `http://localhost:3001`

Login Credentials:

Admin:

- Username: `admin`
- Password: `admin123`

Manager:

- Username: `manager`
- Password: `manager123`

Receptionist:

- Username: `receptionist`
- Password: `recept123`

Step 6: Stop the Server

To stop the server:

```
Get-Process -Name node -ErrorAction SilentlyContinue | Stop-Process -Force
```

□ Technical Concepts

1. **Single Page Application (SPA)**

The frontend is a React SPA where navigation happens on the client-side.

How it works:

- Initial page load fetches index.html
- React Router handles all navigation
- No page reloads on route changes
- Fast, smooth user experience

Implementation:

- React Router v6 for routing
- Express serves index.html for all non-API routes
- Catch-all route: `app.get('*', (req, res) => res.sendFile(...index.html))`

---

2. **REST API Architecture**

Backend provides RESTful API endpoints for CRUD operations.

REST Principles:

- Resource-based URLs (`/api/bookings`, `/api/customers`)
- HTTP methods: GET (read), POST (create), PUT (update), DELETE (delete)
- Stateless communication
- JSON data format

Example:

```
GET    /api/customers    - Get all customers
POST   /api/customers    - Create new customer
PUT    /api/customers/5  - Update customer ID 5
DELETE /api/customers/5  - Delete customer ID 5
```

---

3. **Context API for State Management**

React Context API manages global authentication state.

Benefits:

- No prop drilling
- Centralized authentication logic
- Easy access from any component
- Minimal boilerplate

Implementation:

```
// Create context
const AuthContext = createContext();

// Provider component
<AuthProvider>
  <App />
</AuthProvider>

// Use in components
const { user, login, logout } = useAuth();
```

---

4. **Protected Routes**

Routes are protected with authentication check.

How it works:

1. User tries to access protected route
2. ProtectedRoute component checks if user is logged in
3. If yes, render the component
4. If no, redirect to login page

Code:

```
<ProtectedRoute>
  <Dashboard />
</ProtectedRoute>
```

---

5. **Role-Based UI Rendering**

UI elements shown/hidden based on user permissions.

Implementation:

```
const menuItems = [
  {
    path: '/staff',
    label: 'Staff',
    permission: hasPermission('canManageStaff')
  }
];

const filteredItems = menuItems.filter(item => item.permission);
```

Benefits:

- Cleaner UI
- Better UX
- Security through obscurity
- Less confusion for users

---

6. **Real-time Data Calculation**

Analytics dashboard calculates metrics from live database data.

No Caching:

- Every page load fetches fresh data
- Calculations done in real-time
- Always shows current state

Metrics Calculated:

- Occupancy Rate = (Occupied Rooms / Total Rooms) × 100
- Monthly Revenue = Sum of payments in last 30 days
- Active Bookings = Count of Confirmed/Checked-in status

---

7. **Single Port Deployment**

Both frontend and backend run on same port (3001).

Architecture:

```
Port 3001
├── /api/* → Backend API routes
├── /assets/* → Frontend static files (CSS, JS)
└── /* → index.html (React app)
```

Benefits:

- No CORS issues
- Simplified deployment
- Single URL for entire application
- Production-ready setup

Implementation:

```
// Serve static files
app.use(express.static(path.join(__dirname, '../hotel-management-frontend/dist')));

// API routes
app.use('/api/bookings', bookingsRoutes);

// Catch-all for React Router
app.get('*', (req, res) => {
  res.sendFile(path.join(__dirname, '../hotel-management-frontend/dist', 'index.html'));
});
```

---

8. **Form Validation**

Both client-side and server-side validation for data integrity.

Frontend Validation:

- Required field checks
- Email format validation
- Phone number format
- Date range validation
- Prevents invalid form submission

Backend Validation:

- Duplicate check (email, username, room number)
- Data type validation
- Business logic validation
- SQL injection prevention

---

9. **Error Handling**

Comprehensive error handling at all levels.

Frontend:

- Try-catch blocks for API calls
- User-friendly error messages
- Error state management

Backend:

- Database error handling
- Validation error responses
- 404 for not found
- 500 for server errors

Database:

- Foreign key constraints
- Unique constraints
- Transaction rollback on error

□ User Roles & Permissions Matrix

Feature	Super Admin	Admin	Manager	Receptionist
View Dashboard	□	□	□	□
Manage Staff	□	□	□	□
View Customers	□	□	□	□
Add/Edit Customers	□	□	□	□
Delete Customers	□	□	□	□
Manage Rooms	□	□	□	□ (View only)
View Bookings	□	□	□	□
Create Bookings	□	□	□	□
Edit Bookings	□	□	□	□
Cancel Bookings	□	□	□	□
View Payments	□	□	□	□
Record Payments	□	□	□	□
View Analytics	□	□	□	□
Manage Services	□	□	□	□
Delete Records	□	□	□	□

□ Summary

The Hotel Management System is a comprehensive, full-stack web application that streamlines hotel operations with modern technologies and best practices.

Key Highlights:

- □ Complete CRUD operations for all entities
- □ Role-based access control with 4 user roles
- □ Real-time analytics and reporting
- □ Indian currency (₹) and Karnataka localization
- □ Single-port deployment for easy access
- □ Clean, minimal UI design
- □ Secure authentication system
- □ Production-ready architecture

Technology Stack:

- Frontend: React 18 + Vite + Tailwind CSS
- Backend: Node.js + Express.js
- Database: MySQL
- Deployment: Single port (3001)

Perfect for:

- Small to medium-sized hotels
- Guesthouses
- Resorts
- Boutique hotels

□ Support & Contact

For any questions or support:

- Developer: Shruthana
- Email: shruthanaj@hotel.com
- Date: October 31, 2025

End of Documentation □