Hotel Management System - Software Design Document

1. System Overview

1.1 Purpose

The Hotel Management System (HMS) is a microservices-based application designed to manage hotel operations including room bookings, billing, user management, and reporting. The system supports multiple user roles (Admin, Guest, Manager, Receptionist) with role-based access control.

1.2 Architecture

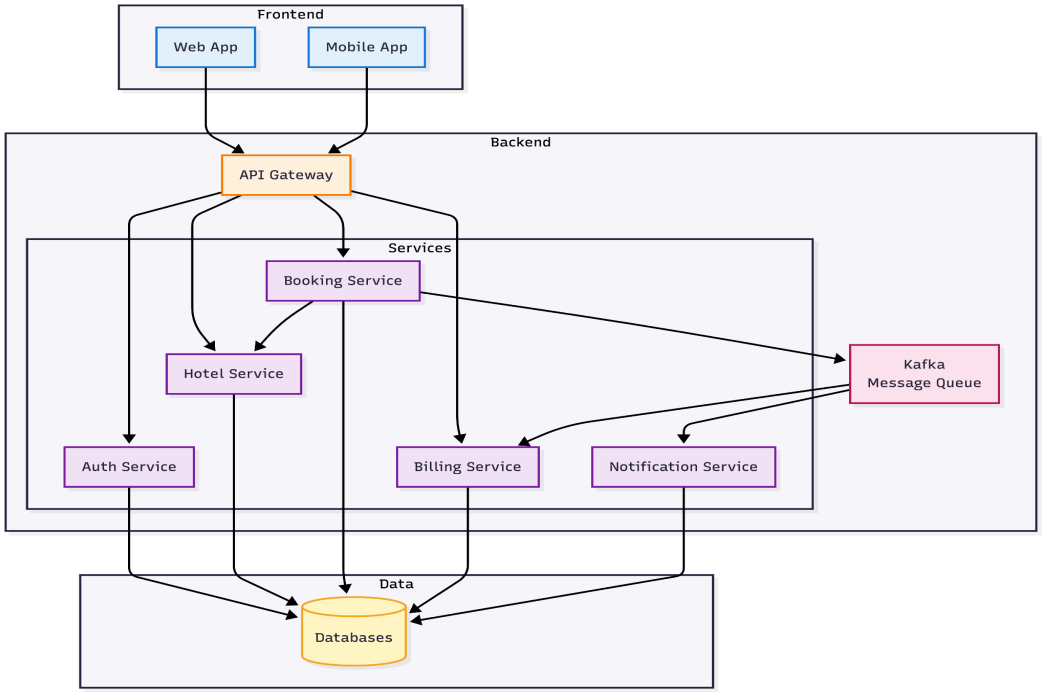

Fig: Architecture Diagram

- **Architecture Pattern**: Microservices with API Gateway
- **Service Discovery**: Netflix Eureka
- **Configuration Management**: Spring Cloud Config Server
- **Message Queue**: Apache Kafka (event-driven communication)
- **Database**: MySQL (separate database per service)
- **Cache**: Redis (for reports service)
- **Authentication**: JWT-based authentication

1.3 Services

1. **Eureka Service** (Port 8761) - Service discovery
2. **Config Server** (Port 8888) - Centralized configuration
3. **API Gateway** (Port 8080) - Single entry point, routing, authentication
4. **Auth Service** (Port 9001) - User management, authentication, authorization
5. **Hotel Service** (Port 9002) - Hotel and room management
6. **Booking Service** (Port 9003) - Booking management, availability checks
7. **Billing Service** (Port 9005) - Bill generation and payment tracking
8. **Notification Service** (Port 9004) - Email/SMS notifications
9. **Reports Service** (Port 9006) - Dashboards and analytics

---

2. User Roles & Access Control

2.1 Roles

- **ADMIN**: System administrator, full access to all hotels and operations
- **GUEST**: Regular users who can make bookings
- **MANAGER**: Hotel manager, bound to a specific hotel
- **RECEPTIONIST**: Front desk staff, bound to a specific hotel

2.2 Access Control Rules

- **ADMIN & GUEST**: Not bound to any hotel (`hotelId = null`)
- **MANAGER & RECEPTIONIST**: Must be bound to a hotel (`hotelId` required)
- Staff users can only access resources for their assigned hotel
- JWT tokens include `hotelId` for staff users

- API Gateway extracts user info and adds headers: `X-User-Id`, `X-User-Role`, `X-Hotel-Id`, `X-User-Username`, `X-User-Public-Id`

---

3. API Endpoints

3.1 Auth Service (Port 9001)

Public Endpoints

Method	Endpoint	Description	Access
POST	`/auth/register/guest`	Register a new guest user	Public
POST	`/auth/login`	User login, returns JWT token	Public
POST	`/auth/activate`	Activate account using activation token	Public

Protected Endpoints

Method	Endpoint	Description	Access
GET	`/auth/me`	Get current user details	Authenticated users
POST	`/auth/change-password`	Change user password	Authenticated users

Admin Endpoints

Method	Endpoint	Description	Access
POST	`/auth/admin/staff`	Create staff user (MANAGER/RECEPTIONIST)	ADMIN only
GET	`/auth/admin/users`	List all users	ADMIN only
PATCH	`/auth/admin/users/{userId}/deactivate`	Deactivate user	ADMIN only

Method	Endpoint	Description	Access
PATCH	`/auth/admin/users/{userId}/activate`	Activate user	ADMIN only
PUT	`/auth/admin/staff/{userId}/hotel-allotment`	Reassign staff to hotel	ADMIN only
GET	`/auth/admin/users/hotel/{hotelId}`	Get users by hotel	ADMIN only
PUT	`/auth/admin/staff/{userId}`	Update staff user	ADMIN only

Internal Endpoints (Feign)

Method	Endpoint	Description	Used By
GET	`/internal/auth/users/{userId}`	Get user by ID	Notification Service, other services

3.2 Hotel Service (Port 9002)

Public Endpoints

Method	Endpoint	Description	Access
GET	`/hotels/search`	Search hotels by city or category	Public
GET	`/hotels/{hotelId}`	Get hotel details	Public
GET	`/hotels/availability/search`	Search room availability	Public
GET	`/hotels/rooms/hotel/{hotelId}`	Get rooms by hotel	Public

Method	Endpoint	Description	Access
GET	`/hotels/rooms/{roomId}`	Get room by ID	Public

Protected Endpoints

Method	Endpoint	Description	Access
GET	`/hotels/my-hotel`	Get staff's assigned hotel	MANAGER, RECEPTIONIST
GET	`/hotels`	Get all hotels	ADMIN only
POST	`/hotels`	Create hotel	ADMIN only
PUT	`/hotels/{hotelId}`	Update hotel	ADMIN (any), MANAGER (own hotel)
GET	`/hotels/{hotelId}/staff`	Get staff for hotel	ADMIN only
POST	`/hotels/{hotelId}/staff`	Create staff user for hotel	ADMIN only
POST	`/hotels/rooms`	Create room	ADMIN (any), MANAGER (own hotel)
PUT	`/hotels/rooms/{roomId}`	Update room	ADMIN (any), MANAGER (own hotel)
DELETE	`/hotels/rooms/{roomId}`	Delete room (soft delete)	ADMIN (any), MANAGER (own hotel)
PATCH	`/hotels/rooms/{roomId}/status`	Update room status	ADMIN, MANAGER, RECEPTIONIST (own hotel)
PATCH	`/hotels/rooms/{roomId}/active`	Activate/deactivate room	ADMIN (any), MANAGER (own hotel)
POST	`/hotels/availability/block`	Block room for dates	ADMIN (any), MANAGER (own hotel)
POST	`/hotels/availability/unblock`	Unblock room for dates	ADMIN (any), MANAGER (own hotel)

Internal Endpoints (Feign)

Method	Endpoint	Description	Used By
GET	`/internal/hotels/{hotelId}`	Get hotel by ID	Booking Service, Notification Service
GET	`/internal/hotels/{hotelId}/rooms`	Get rooms by hotel	Booking Service
GET	`/internal/hotels/rooms/{roomId}`	Get room by ID	Booking Service

3.3 Booking Service (Port 9003)

Public Endpoints

Method	Endpoint	Description	Access
GET	`/bookings/check-availability`	Check room availability	Public

Protected Endpoints

Method	Endpoint	Description	Access
POST	`/bookings`	Create booking	GUEST, ADMIN
POST	`/bookings/walk-in`	Create walk-in booking	MANAGER, RECEPTIONIST, ADMIN
GET	`/bookings/{bookingId}`	Get booking by ID	Owner, ADMIN, MANAGER (own hotel)
GET	`/bookings/my-bookings`	Get user's bookings	Authenticated users
GET	`/bookings/hotel/{hotelId}`	Get bookings by hotel	MANAGER, RECEPTIONIST, ADMIN (own hotel)

Method	Endpoint	Description	Access
GET	`/bookings`	Get all bookings	ADMIN only
POST	`/bookings/{bookingId}/confirm`	Confirm booking	ADMIN, MANAGER
POST	`/bookings/{bookingId}/cancel`	Cancel booking	Owner, ADMIN
POST	`/bookings/{bookingId}/check-in`	Check-in guest	MANAGER, RECEPTIONIST, ADMIN (own hotel)
POST	`/bookings/{bookingId}/check-out`	Check-out guest	MANAGER, RECEPTIONIST, ADMIN (own hotel)
GET	`/bookings/hotel/{hotelId}/today-checkins`	Get today's check-ins	MANAGER, RECEPTIONIST, ADMIN (own hotel)
GET	`/bookings/hotel/{hotelId}/today-checkouts`	Get today's check-outs	MANAGER, RECEPTIONIST, ADMIN (own hotel)

Internal Endpoints (Feign)

Method	Endpoint	Description	Used By
GET	`/internal/bookings/{bookingId}`	Get booking by ID	Billing Service
POST	`/internal/bookings/{bookingId}/confirm`	Confirm booking	Billing Service

3.4 Billing Service (Port 9005)

Protected Endpoints

Method	Endpoint	Description	Access
GET	`/bills/booking/{bookingId}`	Get bill by booking ID	Owner, ADMIN
POST	`/bills/generate/{bookingId}`	Manually generate bill	ADMIN only
POST	`/bills/{billId}/mark-paid`	Mark bill as paid	ADMIN only
GET	`/bills/my-payments`	Get user's payment history	Authenticated users
GET	`/bills/payments`	Get all payments	ADMIN only

Note: Bills are automatically generated when a booking is confirmed (via Kafka event).

3.5 Reports Service (Port 9006)

Protected Endpoints

Method	Endpoint	Description	Access
GET	`/reports/dashboard/manager`	Manager dashboard metrics	MANAGER, RECEPTIONIST (own hotel)
GET	`/reports/dashboard/admin`	Admin dashboard metrics	ADMIN only
GET	`/reports/dashboard/admin?hotelId={hotelId}`	Admin dashboard for specific hotel	ADMIN only
GET	`/reports/hotels`	Get hotels list	ADMIN, MANAGER

Dashboard Metrics:

- Total Revenue (all-time)
- Today Revenue
- Total Bookings (all-time)
- Today Bookings
- Check-ins Today
- Check-outs Today
- Available Rooms Today
- Average Rating
- Today Revenue by Hour (0-23)
- Today Bookings by Source (PUBLIC/WALK_IN) by Hour

3.6 Notification Service (Port 9004)

Note: Notification Service is event-driven and does not expose REST endpoints. It listens to Kafka events:

- `booking-created` - Schedules check-in reminders
- `booking-confirmed` - Sends confirmation email
- `guest-checked-in` - Sends welcome email
- `checkout-completed` - Sends thank-you and feedback emails

---

4. Database Schema

4.1 Auth Service Database (`hms_auth_db`)

User Table

```
CREATE TABLE user (
    id BIGINT PRIMARY KEY AUTO_INCREMENT,
    public_user_id VARCHAR(255) UNIQUE,
    username VARCHAR(50) UNIQUE NOT NULL,
    full_name VARCHAR(255) NOT NULL,
    email VARCHAR(255) UNIQUE NOT NULL,
    password VARCHAR(255) NOT NULL,
    password_last_changed_at DATETIME,
```

```
    role ENUM('ADMIN', 'GUEST', 'MANAGER', 'RECEPTIONIST') NOT NULL,
    hotel_id BIGINT NULL,
    enabled BOOLEAN DEFAULT TRUE,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    updated_at DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP
);
```

UserHotelAssignment Table

```
CREATE TABLE user_hotel_assignment (
    id BIGINT PRIMARY KEY AUTO_INCREMENT,
    user_id BIGINT UNIQUE NOT NULL,
    hotel_id BIGINT NOT NULL,
    FOREIGN KEY (user_id) REFERENCES user(id),
    FOREIGN KEY (hotel_id) REFERENCES hotel(id) -- in hotel service DB
);
```

ActivationToken Table

```
CREATE TABLE activation_token (
    id BIGINT PRIMARY KEY AUTO_INCREMENT,
    user_id BIGINT NOT NULL,
    token VARCHAR(255) UNIQUE NOT NULL,
    expires_at DATETIME NOT NULL,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES user(id)
);
```

4.2 Hotel Service Database (`hms_hotel_db`)

Hotel Table

```
CREATE TABLE hotel (
    id BIGINT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(255) NOT NULL,
    category ENUM('LUXURY', 'BUDGET', 'BUSINESS', 'BOUTIQUE', 'RESORT') NOT NULL,
    description TEXT,
    city ENUM('MUMBAI', 'DELHI', 'BANGALORE', 'KOLKATA', 'CHENNAI', 'HYDERABAD',
'PUNE', 'JAIPUR') NOT NULL,
```

```
    address VARCHAR(500) NOT NULL,
    state ENUM('MAHARASHTRA', 'DELHI', 'KARNATAKA', 'WEST_BENGAL', 'TAMIL_NADU',
'TELANGANA', 'RAJASTHAN') NOT NULL,
    country VARCHAR(100) DEFAULT 'India',
    pincode VARCHAR(10) NOT NULL,
    email VARCHAR(255) UNIQUE NOT NULL,
    contact_number VARCHAR(20) UNIQUE NOT NULL,
    star_rating INT CHECK (star_rating BETWEEN 1 AND 5),
    amenities TEXT,
    image_url VARCHAR(500),
    status ENUM('ACTIVE', 'INACTIVE') DEFAULT 'ACTIVE',
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    updated_at DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP
);
```

Room Table

```
CREATE TABLE room (
    id BIGINT PRIMARY KEY AUTO_INCREMENT,
    hotel_id BIGINT NOT NULL,
    room_number VARCHAR(50) NOT NULL,
    room_category ENUM('DELUXE', 'STANDARD', 'SUITE', 'EXECUTIVE', 'FAMILY') NOT
NULL,
    price_per_night DECIMAL(10,2) NOT NULL,
    max_occupancy INT NOT NULL,
    floor_number INT,
    bed_type VARCHAR(100),
    room_size INT,
    amenities TEXT,
    description TEXT,
    status ENUM('AVAILABLE', 'OCCUPIED', 'MAINTENANCE', 'OUT_OF_SERVICE')
DEFAULT 'AVAILABLE',
    is_active BOOLEAN DEFAULT TRUE,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    updated_at DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
    FOREIGN KEY (hotel_id) REFERENCES hotel(id),
    UNIQUE KEY unique_room_per_hotel (hotel_id, room_number)
);
```

RoomAvailability Table

```
CREATE TABLE room_availability (
    id BIGINT PRIMARY KEY AUTO_INCREMENT,
    hotel_id BIGINT NOT NULL,
    room_id BIGINT NOT NULL,
    availability_date DATE NOT NULL,
    status ENUM('AVAILABLE', 'RESERVED', 'BLOCKED') DEFAULT 'AVAILABLE',
    source VARCHAR(50),
    reason TEXT,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    updated_at DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
    FOREIGN KEY (hotel_id) REFERENCES hotel(id),
    FOREIGN KEY (room_id) REFERENCES room(id),
    UNIQUE KEY unique_room_date (room_id, availability_date)

);
```

4.3 Booking Service Database (`hms_booking_db`)

Booking Table

```
CREATE TABLE booking (
    id BIGINT PRIMARY KEY AUTO_INCREMENT,
    user_id BIGINT NOT NULL,
    hotel_id BIGINT NOT NULL,
    room_id BIGINT NOT NULL,
    room_number VARCHAR(50),
    room_type VARCHAR(50),
    check_in_date DATE NOT NULL,
    check_out_date DATE NOT NULL,
    total_amount DECIMAL(10,2) NOT NULL,
    status ENUM('CREATED', 'CONFIRMED', 'CHECKED_IN', 'CHECKED_OUT',
'CANCELLED') DEFAULT 'CREATED',
    booking_source ENUM('PUBLIC', 'WALK_IN', 'PHONE') DEFAULT 'PUBLIC',
    guest_name VARCHAR(255),
    guest_email VARCHAR(255),
    guest_phone VARCHAR(20),
    number_of_guests INT NOT NULL,
```

```
    number_of_nights INT NOT NULL,
    special_requests TEXT,
    guest_details TEXT,
    cancellation_reason TEXT,
    cancelled_at DATETIME,
    checked_in_at DATETIME,
    checked_out_at DATETIME,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    updated_at DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP
);
```

ScheduledReminder Table

```
CREATE TABLE scheduled_reminder (
    id BIGINT PRIMARY KEY AUTO_INCREMENT,
    booking_id BIGINT NOT NULL,
    user_id BIGINT NOT NULL,
    hotel_id BIGINT NOT NULL,
    reminder_type VARCHAR(100) NOT NULL,
    scheduled_date DATE NOT NULL,
    check_in_date DATE NOT NULL,
    guest_email VARCHAR(255),
    guest_name VARCHAR(255),
    sent BOOLEAN DEFAULT FALSE,
    cancelled BOOLEAN DEFAULT FALSE,
    sent_at DATETIME,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (booking_id) REFERENCES booking(id)

);
```

4.4 Billing Service Database (`hms_billing_db`)

Bill Table

```
CREATE TABLE bill (
    id BIGINT PRIMARY KEY AUTO_INCREMENT,
    booking_id BIGINT UNIQUE NOT NULL,
```

```
    user_id BIGINT NOT NULL,
    hotel_id BIGINT NOT NULL,
    room_id BIGINT NOT NULL,
    check_in_date DATE NOT NULL,
    checkout_date DATE NOT NULL,
    total_amount DECIMAL(10,2) NOT NULL,
    status ENUM('PENDING', 'PAID') DEFAULT 'PENDING',
    bill_number VARCHAR(255) UNIQUE NOT NULL,
    generated_at DATETIME NOT NULL,
    paid_at DATETIME,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    updated_at DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP
);
```

Payment Table

```
CREATE TABLE payment (
    id BIGINT PRIMARY KEY AUTO_INCREMENT,
    bill_id BIGINT NOT NULL,
    booking_id BIGINT NOT NULL,
    user_id BIGINT NOT NULL,
    amount DECIMAL(10,2) NOT NULL,
    payment_method VARCHAR(50),
    transaction_id VARCHAR(255),
    payment_reference VARCHAR(255),
    notes TEXT,
    paid_by VARCHAR(255),
    paid_at DATETIME NOT NULL,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (bill_id) REFERENCES bill(id)
);
```

---

5. Business Rules

5.1 User Management

1. **Guest Registration**: Guests can self-register via `/auth/register/guest`

2. **Staff Creation**: Only ADMIN can create staff users (MANAGER/RECEPTIONIST)
3. **Hotel Binding**: MANAGER and RECEPTIONIST must be bound to a hotel at creation
4. **Account Activation**: Staff users receive activation tokens (for future email activation)
5. **Password Policy**: Passwords are encrypted using BCrypt, never stored in plain text

5.2 Hotel Management

1. **Hotel Creation**: Only ADMIN can create hotels
2. **Hotel Status**: Hotels can be ACTIVE or INACTIVE
3. **Room Uniqueness**: Room numbers must be unique within a hotel
4. **Soft Delete**: Rooms are soft-deleted (`isActive = false`) instead of hard delete
5. **Room Status**: Rooms can be AVAILABLE, OCCUPIED, MAINTENANCE, or OUT_OF_SERVICE

5.3 Booking Management

1. **Booking Creation**: GUEST can create bookings via system, MANAGER/ADMIN can create walk-in bookings
2. **Booking Status Flow**: CREATED → CONFIRMED → CHECKED_IN → CHECKED_OUT (or CANCELLED)
3. **Availability Check**: System checks room availability before booking creation
4. **Double Booking Prevention**: System prevents double booking of same room for overlapping dates
5. **Check-in Rules**: Only CONFIRMED bookings can be checked in
6. **Cancellation**: GUEST can cancel own bookings, ADMIN can cancel any booking. Cannot cancel CHECKED_IN or CHECKED_OUT bookings
7. **Booking Source**: Automatically tagged as PUBLIC (guest booking) or WALK_IN (staff booking)

5.4 Billing & Payment

1. **Automatic Bill Generation**: Bills are automatically generated when booking is confirmed (via Kafka event)
2. **One Booking = One Bill**: Each booking generates exactly one bill
3. **Bill Status**: Bills start as PENDING, change to PAID when marked as paid
4. **Payment Records**: Payment records are append-only (immutable)
5. **Payment Authorization**: Only ADMIN can mark bills as paid
6. **User Access**: Users can only view their own bills/payments (unless ADMIN)

5.5 Availability Management

1. **Availability Calculation**: Booking Service calculates availability by checking:
    - Existing bookings for date range
    - Blocked rooms (via RoomAvailability table)
    - Room status (MAINTENANCE, OUT_OF_SERVICE)
2. **Room Blocking**: ADMIN and MANAGER can block rooms for maintenance or other reasons
3. **Date Range Validation**: Check-out date must be after check-in date

5.6 Authorization Rules

1. **Context-Aware Access**: Staff users can only access resources for their assigned hotel
2. **Admin Override**: ADMIN can access all hotels and operations
3. **Resource Ownership**: Users can view/modify their own bookings (unless ADMIN/MANAGER)
4. **Hotel-Specific Operations**: Staff operations are scoped to their hotel (from JWT `X-Hotel-Id`)

5.7 Notification Rules

1. **Booking Created**: Schedules check-in reminder (24 hours before check-in)
2. **Booking Confirmed**: Sends confirmation email to guest
3. **Guest Checked In**: Sends welcome email, cancels reminder
4. **Guest Checked Out**: Sends thank-you email and feedback request

5.8 Reporting Rules

1. **Manager Dashboard**: Shows metrics only for manager's assigned hotel
2. **Admin Dashboard**: Shows aggregated metrics for all hotels (or specific hotel if `hotelId` provided)
3. **Read-Only Service**: Reports service is read-only, no data modifications
4. **Data Aggregation**: Metrics calculated from booking, billing, and hotel databases

---

6. Integration Points

6.1 Service-to-Service Communication

- **Feign Clients**: Synchronous HTTP calls between services
  - Booking Service → Hotel Service (get hotel/room details)
  - Hotel Service → Auth Service (create staff users)
  - Billing Service → Booking Service (confirm bookings)
  - Notification Service → Auth Service (get user details)
  - Notification Service → Hotel Service (get hotel details)

6.2 Event-Driven Communication (Kafka)

- **Topics**:
  - `booking-created`: Published when booking is created
  - `booking-confirmed`: Published when booking is confirmed
  - `guest-checked-in`: Published when guest checks in
  - `checkout-completed`: Published when guest checks out
- **Consumers**:
  - **Notification Service**: Listens to all booking events
  - **Billing Service**: Listens to `booking-confirmed` event

6.3 API Gateway

- **Routing**: Routes requests to appropriate services
- **Authentication**: Validates JWT tokens
- **Header Injection**: Adds user context headers (`X-User-Id`, `X-User-Role`, `X-Hotel-Id`, etc.)
- **Port**: 8080 (all external requests go through gateway)

6.4 Service Discovery

- **Eureka Server**: All services register with Eureka
- **Service Lookup**: Services use Eureka to discover other services
- **Port**: 8761

6.5 Configuration Management

- **Config Server**: Centralized configuration management
- **Git Backend**: Configuration stored in Git repository
- **Port**: 8888

---

7. Technology Stack

7.1 Backend

- **Framework**: Spring Boot 3.x
- **Language**: Java
- **Build Tool**: Maven
- **Service Discovery**: Netflix Eureka
- **API Gateway**: Spring Cloud Gateway
- **Configuration**: Spring Cloud Config
- **Messaging**: Apache Kafka
- **Database**: MySQL 8.0
- **Cache**: Redis
- **ORM**: JPA/Hibernate

7.2 Security

- **Authentication**: JWT (JSON Web Tokens)
- **Password Encryption**: BCrypt
- **Authorization**: Role-based access control (RBAC)

7.3 Communication

- **Synchronous**: REST APIs, Feign Clients
- **Asynchronous**: Apache Kafka

---

8. Deployment Architecture

8.1 Service Ports

Service	Port
Eureka Service	8761
Config Server	8888
API Gateway	8080
Auth Service	9001
Hotel Service	9002
Booking Service	9003
Notification Service	9004
Billing Service	9005
Reports Service	9006

8.2 Database Ports

Component	Port
MySQL	3306

8.3 External Services

Component	Port
Kafka	9092
Redis	6379

---

9. Error Handling

9.1 Common Error Codes

- **400 Bad Request**: Invalid input parameters
- **401 Unauthorized**: Missing or invalid JWT token
- **403 Forbidden**: Insufficient permissions
- **404 Not Found**: Resource not found
- **409 Conflict**: Resource conflict (e.g., duplicate booking)
- **500 Internal Server Error**: Server-side errors

9.2 Error Response Format

```
{

  "success": false,

  "message": "Error message",

  "error": "Error type",

  "timestamp": "2026-01-07T10:30:00"

}
```

---

10. Future Enhancements

1. **Payment Gateway Integration**: Integrate with payment gateways for online payments
2. **Review/Rating System**: Allow guests to rate and review hotels
3. **Loyalty Program**: Implement points and rewards system
4. **Advanced Analytics**: More detailed reporting and analytics
5. **Mobile App**: Native mobile applications
6. **Multi-language Support**: Support for multiple languages
7. **Real-time Notifications**: WebSocket-based real-time notifications
8. **Inventory Management**: Advanced inventory and supply chain management

11. Appendix

11.1 Enums

User Role

- ADMIN
- GUEST
- MANAGER
- RECEPTIONIST

Hotel Category

- LUXURY
- BUDGET
- BUSINESS
- BOUTIQUE
- RESORT

Hotel Status

- ACTIVE
- INACTIVE

Room Category

- DELUXE
- STANDARD
- SUITE
- EXECUTIVE
- FAMILY

Room Status

- AVAILABLE
- OCCUPIED
- MAINTENANCE
- OUT_OF_SERVICE

Booking Status

- CREATED
- CONFIRMED
- CHECKED_IN
- CHECKED_OUT
- CANCELLED

Booking Source

- PUBLIC
- WALK_IN
- PHONE

Bill Status

- PENDING
- PAID

Payment Method

- CASH
- CARD
- UPI

Availability Status

- AVAILABLE
- RESERVED
- BLOCKED

---

12. Contact & Support

For technical questions or issues, please refer to individual service README files:

- `AUTH_SERVICE_README.md`
- `HOTEL_SERVICE_README.md`
- `BOOKING_SERVICE_README.md`
- `BILLING_SERVICE_README.md`
- `REPORTS_SERVICE_README.md`

- `NOTIFICATION_SERVICE_README.md`

Document Version: 1.0
Last Updated: January 2026
Author: Hotel Management System Development Team

13. System Constraints

13.1 Performance Constraints

- **Latency**: API Gateway response time for critical endpoints (booking creation, check-in) must be below 500ms under expected load.
- **Throughput**: System must support up to 50 concurrent transactions per second (TPS) for booking operations.
- **Scalability**: All stateless microservices (Hotel, Booking, Billing, Reports, Notification) must be horizontally scalable.

13.2 Security Constraints

- **JWT Expiration**: JWT tokens must have a short expiration time (e.g., 15 minutes) with support for refresh tokens (to be implemented in future phase).
- **Sensitive Data**: PII (Personally Identifiable Information) must be encrypted at rest (e.g., customer passwords using BCrypt).
- **Role-Based Access Control**: Strict enforcement of RBAC must be implemented at the API Gateway and within each service.

13.3 Operational Constraints

- **Uptime**: Target 99.9% availability for core services (Auth, Hotel, Booking).
- **Monitoring**: Centralized logging (e.g., ELK stack) and monitoring (e.g., Prometheus/Grafana) are required for all services.
- **Backup/Recovery**: Daily database backups with a defined recovery point objective (RPO) and recovery time objective (RTO).

13.4 Technology Constraints

- **Java/Spring Boot**: All backend services must be developed using Java and Spring Boot.
- **MySQL Compatibility**: Database schemas must be compatible with MySQL 8.0.
- **Kafka Reliability**: Messages must be handled with at-least-once delivery semantics.