System Architecture

Hotel Booking System — Backend Architecture

Version: 1.0.0

Last Updated: November 9, 2025

Author: Aswinnath TE

1 | Overview

The Hotel Booking System (HBS) backend is a **layered asynchronous architecture** built with **FastAPI**, designed for modular scalability and maintainable domain isolation.

It integrates **PostgreSQL** for structured relational data, **MongoDB** for audit and content records, and **Redis** for permission caching and session storage.

Attribute	Specification
Framework	FastAPI (Python, Async)
Architecture Pattern	Layered (Clean) Architecture
Databases	PostgreSQL (ACID), MongoDB (Document), Redis (Cache)
ORM	SQLAlchemy Async
Security Layer	JWT Authentication, RBAC Authorization
Validation	Pydantic v2 Models

2 | Core Architecture Layout

```
Client → Middleware → API Routes → Services → Repositories → Databases
```

Layer Description

Layer	Responsibility
Middleware	Global policies such as CORS, request logging, and exception handling
API Routes	Defines feature-specific endpoints and injects dependencies
Service Layer	Implements business rules and orchestration logic
Data Access Layer (CRUD)	Manages database interactions using ORM abstractions
Database Layer	Handles relational and document-based persistence
Security Layer	Centralized token verification and permission resolution
Audit Layer	Writes activity and change logs to MongoDB

3 | High-Level System Architecture

```
                    ┌────────────────────────────────┐
                    │        HTTP API Requests        │
                    │ (Web, Mobile, Integrations)     │
                    └───────────────┬────────────────┘
                                    ↓
┌──────────────────────────────────────────────────────────────┐
│                    FastAPI Backend Core                      │
│                                                              │
│  [1] Middleware Layer                                        │
│       - CORS configuration                                   │
│       - Request logging middleware                           │
│       - Centralized error handler                            │
│                                                              │
│  [2] API Routes Layer                                        │
│       - Modular routers (bookings, rooms, payments, etc.)    │
│       - Dependency injection                                 │
│                                                              │
│  [3] Service Layer                                           │
│       - Domain logic, pricing, offers, and validation rules  │
│       - Executes use-case workflows                          │
│                                                              │
│  [4] Data Access Layer                                       │
│       - SQLAlchemy async queries for PostgreSQL              │
│       - Redis client for permission and session caching      │
│                                                              │
│  [5] Database Layer                                          │
│       - PostgreSQL: relational core                          │
│       - MongoDB: audit + content                             │
│       - Redis: session + permission cache                    │
└──────────────────────────────────────────────────────────────┘
```

4 | Functional Segmentation

Domain	Description
Authentication	JWT token creation, validation, and refresh

Domain	Description
Authorization	RBAC role and permission management
Bookings	Reservation creation, update, and cancellation
Payments	Records payment status and links to bookings
Refunds	Refund request and approval flow
Rooms	Room CRUD, amenities, and availability management
Offers	Discount and validity management
Reviews	Customer reviews and admin responses
Issues	Support ticket creation and updates
Notifications	User alerts stored per event
Content Management	CMS for static content pages
Audit Logs	Logs system actions and changes in MongoDB

5 | Data Persistence Architecture

PostgreSQL (Relational Core)

- Manages structured entities: `users` , `rooms` , `bookings` , `payments` , `refunds` , `offers` , `issues` , `roles` , and `permissions` .
- Strong foreign key relationships and 3NF schema.
- Indexed for query performance on key fields.
- Soft deletion via `is_deleted` and timestamp columns.

MongoDB (Document Store)

- Stores audit logs and content documents.
- Collections: `audit_logs` , `booking_logs` , `content_docs` , `backup_data_collections` .
- Append-only write pattern for audit reliability.
- Linked to SQL entities via foreign identifiers.

Redis (Cache Layer)

- Stores user permission mappings and session information.
- TTL-based key expiration for session cleanup.
- Used for fast authorization checks and basic caching.

6 | Internal Request Flow

```
1. Client sends HTTPS request with JWT token.
2. Middleware applies CORS, logs request metadata, and catches errors.
3. JWT decoded and verified; user permissions fetched.
4. Route handler invokes corresponding service function.
5. Service executes business logic and coordinates repository operations.
6. Repository writes to PostgreSQL and logs activity to MongoDB.
7. Redis updated with any permission or session changes.
8. Response validated via Pydantic and returned as JSON.
```

7 | Architectural Principles

Principle	Implementation
Layered Separation	Routes, services, and data access layers decoupled.
Single Source of Truth	PostgreSQL maintains authoritative records.
Async Processing	Fully asynchronous stack with SQLAlchemy.
Consistency	Strong in PostgreSQL, eventual in MongoDB.
Auditability	Key events logged via MongoDB.
Caching	Permission and session caching via Redis.
Extensibility	Modular route and service structure for new features.

8 | Key Architectural Benefits

Benefit	Description
Modular Layers	Decoupled architecture enabling independent feature evolution.
Multi-Database Model	SQL for transactional data, Mongo for audit, Redis for speed.
High Throughput	Async I/O and efficient query access patterns.
Security-Centric	JWT and RBAC with middleware-driven enforcement.
Audit Ready	Persistent operational logs for traceability.
Maintainable	Structured directory layout with consistent service boundaries.

9 | Component Summary

Component	Description
FastAPI Core	Central async application handling all HTTP traffic.
Middleware	Manages request lifecycle and error boundaries.
Routes	Domain-based routers providing endpoint segregation.
Services	Business rules and workflow orchestration layer.
CRUD Modules	Database abstraction layer using ORM.
PostgreSQL	Primary structured data store.
MongoDB	Secondary document and audit store.
Redis	Lightweight cache for session and permissions.