Amrutam Telemedicine Backend — Architecture Documentation

---

1. System Overview

Project Introduction

Amrutam Telemedicine Backend is a modular RESTful backend system designed to support doctor availability management and secure appointment booking workflows. The system emphasizes scalability, reliability, and concurrency-safe operations while maintaining strong security and observability practices.

The platform supports multiple user roles — patients, doctors, and administrators — through authenticated APIs handling scheduling, booking, and administrative analytics.

Primary Goals

Scalability
Stateless APIs and efficient database design allow horizontal scaling to support increasing consultation traffic.

Reliability
Critical workflows use database transactions and concurrency control mechanisms to maintain data integrity during simultaneous requests.

Secure Booking
JWT authentication and Role-Based Access Control (RBAC) ensure only authorized users perform sensitive actions. Booking operations are protected against duplicate submissions.

Auditability
All critical actions are recorded in audit logs to enable traceability and operational monitoring.

---

2. High-Level Architecture

Architecture Overview

The system follows a modular monolithic architecture separating business domains while keeping deployment simple.

Client Applications (Web / Mobile)

↓

Express API Layer

↓

Auth Module | Doctor Module | Booking Module

Admin Module | Audit Module

↓

PostgreSQL Database

↓

Docker Runtime

Modular Design

Responsibilities are divided into independent modules:

- **Authentication Module** — user signup, login, and JWT validation.
- **Doctor Module** — doctor profiles and availability slot management.
- **Booking Module** — appointment lifecycle handling.
- **Admin Module** — analytics endpoints.
- **Audit Module** — system activity tracking.

This separation improves maintainability and future extensibility.

Stateless API Design

The backend operates using stateless APIs where authentication context is passed via JWT tokens in every request. This enables multiple API instances to run concurrently behind a load balancer.

RBAC Middleware

Role-based authorization ensures controlled access:

- Doctors manage availability slots.
- Patients book appointments.
- Admins access analytics endpoints.

Authorization checks are enforced via middleware before executing business logic.

Containerized Deployment

The application is containerized using Docker, ensuring consistent environments and reproducible deployments across systems. Docker Compose orchestrates API and PostgreSQL services together.

---

3. Booking Architecture (Concurrency-Safe Design)

The booking workflow is the most critical system component because it must prevent double booking during concurrent requests.

Transactional Booking Flow

Booking requests execute inside a PostgreSQL transaction where:

- JWT authentication is verified.
- Idempotency key validation prevents duplicate processing.
- Slot row is locked using `SELECT FOR UPDATE`.
- Booking record is created atomically.
- Slot is marked as booked.
- Audit log entry is recorded.
- Transaction commits successfully.

If any step fails, the transaction rolls back automatically.

Row-Level Locking

The system uses:

```
SELECT ... FOR UPDATE
```

This locks the selected availability slot during booking, preventing concurrent modifications and ensuring only one successful booking per slot.

Idempotency Keys

Idempotency keys allow safe request retries. If a booking request is repeated due to network failure, the system returns the stored response instead of creating duplicate records.

Audit Logging

Each booking action records:

- user performing the action
- action type
- affected entity
- timestamp

This provides complete operational traceability.

---

4. Scalability and Observability

Horizontal Scaling

Because APIs are stateless, multiple backend instances can run behind a load balancer, enabling the system to support high consultation volumes.

Load Balancer

↓

API Instance 1

API Instance 2

API Instance N

Caching Strategy

Frequently accessed data such as doctor listings, availability slots, and analytics responses can be cached using Redis to reduce database load and improve response latency.

Database Optimization

Scalability is supported through:

- indexed foreign keys
- optimized booking queries
- future partitioning by booking date

Backup and Disaster Recovery

Daily PostgreSQL backups ensure data recovery capability. In failure scenarios, a new database instance can be restored from backups and API containers reconnect automatically, minimizing downtime.

Logging and Monitoring (Observability)

Current observability includes:

- HTTP request logging using Morgan
- security headers via Helmet
- persistent audit logging

Future production extensions include Prometheus metrics collection, Grafana dashboards, and distributed tracing.

Key monitored metrics include API latency, booking success rate, error frequency, and system uptime.

Conclusion

The Amrutam Telemedicine Backend demonstrates a production-focused backend architecture emphasizing security, concurrency safety, scalability, and operational visibility. Transactional database design, RBAC enforcement, auditability, and containerized deployment ensure reliable handling of real-world telemedicine workflows while remaining scalable for future growth.