**Non-Functional Requirements – Design and Architecture Solution**

**🔁 1. Transactional Scenarios & Design Decisions**

**🧾 Booking Flow (Read & Write consistency)**

* **Atomic booking operation** using a **Saga pattern**:
  + Step 1: Lock seats in cache (e.g., Redis) with TTL.
  + Step 2: Persist booking in DB (PostgreSQL or DynamoDB with conditional writes).
  + Step 3: Trigger async event to payment service (Kafka or SQS).
  + Step 4: On payment success, confirm booking; on failure, rollback seat lock.

**🚫 Failure Handling**

* Idempotency keys in booking APIs to prevent double charges.
* Seat lock with expiration ensures deadlock-free availability recovery.

**🏢 2. Integration with Theatre IT Systems & New Theatres**

**🧩 Adapter-Based Design**

* **Legacy theatres**: CSV uploads, flat-file exchange via SFTP.
* **Modern theatres**: REST or SOAP APIs for real-time sync.

**🔄 Synchronization Layer**

* Background job (Spring Batch or Apache Camel) to pull inventory updates hourly/daily.
* API gateway to manage different integration contracts.

**🌐 Localization**

* Movies tagged with language, region, and genre metadata.
* Use i18n libraries (like ICU or Spring Messages) for multilingual support.

**🌍 3. Scalability & Availability (99.99%)**

| **Concern** | **Solution** |
| --- | --- |
| **Scalability** | Stateless microservices behind ALB + Auto Scaling |
| **Storage** | Read-heavy replicas (PostgreSQL + Read Replicas) |
| **Cache** | Redis/Memcached for seat layouts, movie lists |
| **Global Reach** | Multi-region AWS deployment with Route53 |
| **Availability** | Load-balanced, HA setup + RDS Multi-AZ + retries |

**Tools:**

* **Cloud**: AWS (or GCP/Azure equivalent)
* **Resilience**: Circuit breakers (Resilience4j), retries, bulkheads
* **Observability**: Prometheus, Grafana, ELK, AWS CloudWatch

**💳 4. Payment Gateway Integration**

**Payment Gateway Options**

* Razorpay, PayU, Stripe (PCI-DSS compliant)

**Flow:**

* Async payment confirmation via webhook or polling
* Use **JWT tokens + payment callback URL** to link session
* Support for UPI, cards, wallets

**Security:**

* Never store card data; only payment token and reference ID

**💰 5. Monetization Model**

| **Method** | **Notes** |
| --- | --- |
| **Commission per ticket** | Charge 2–5% from theatres |
| **Subscription tiers for B2B** | Premium listing, analytics dashboard |
| **Ad placements** | Sponsored banners, movie promotions |
| **Offer partnerships** | Co-branded offers with banks, UPI |

**🔐 6. OWASP Top 10 Threat Mitigation**

| **OWASP Threat** | **Mitigation Strategy** |
| --- | --- |
| Injection | Use ORM (JPA), PreparedStatements |
| Broken Auth | JWT with short TTL, Refresh tokens |
| Sensitive Data Exposure | HTTPS, Vault for secrets |
| XML External Entities (XXE) | Disable DTD in XML parsers |
| Broken Access Control | Role-based access control |
| Security Misconfig | CI/CD secrets scanning, WAF |
| XSS/CSRF | Input validation, CSRF tokens |
| Insecure Deserialization | Avoid serialized user input |
| Components with Known Vulns | OWASP Dependency Check in CI/CD |
| Logging & Monitoring | Centralized logs + alerting (ELK, Prometheus) |

**📜 7. Compliance Requirements**

| **Compliance Area** | **Compliance Solution** |
| --- | --- |
| **User Data Privacy** | GDPR (right to forget, consent banner) |
| **Payment Data** | PCI-DSS via compliant gateway only |
| **Hosting & Access** | ISO 27001 cloud providers, IAM roles |
| **Data Retention** | Define TTL for booking, audit logs |
| **Auditability** | Event logging, trace ID with each API |

**📌 Summary Diagram**Booking Workflow → Seat Lock (Redis) → Booking (DB) → Payment → Confirm / Fail → Notification (Kafka)

* All services stateless and scalable
* Uses CQRS for separation of read/write flows in future