**Non-functional requirements**

**1. Transactional Scenarios and Design Decisions**

**Scenario 1: Booking a Movie Ticket (B2C)**:

The user picks the movie, theatre, date, and seat.

Payment is processed, and the ticket is confirmed.

Kafka (or event-driven design) helps the system communicate between services without them being tightly connected.

If the payment fails, the reservation is cancelled to keep things consistent (ROLLBACK).

**Scenario 2: Booking a Movie Ticket (B2B)**:

Theatre partners can book tickets for customers.

Once the payment is completed, they receive a confirmation.

**Cancellations/Refunds**:

Users can cancel tickets. If the cancellation is successful, a refund is processed.

**Design Decisions**:

**Event-Driven Architecture**: Use Kafka or another message broker to send messages between services (like booking, payment, and ticket confirmation) without making them wait for each other, allowing processes to run independently.

**Saga Pattern**: A way to handle complex transactions that span multiple services. If something goes wrong (e.g., payment failure), it can automatically undo previous actions to maintain consistency.

**Two-Phase Commit or Distributed Transactions**: A method used for critical services (like payment) to ensure everything happens correctly. It makes sure all parts of a transaction either succeed together or fail together, maintaining consistency across multiple systems.

**2. Integrating with Theatres Having Existing IT Systems and New Theatres (Localization)**

**Design Approach**:

Use **API-First Design**: Expose well-documented APIs for integrating with external theatre systems.

**Adapter Pattern**: Implement adapters for integrating legacy theatre systems and new systems. This allows flexibility for different IT architectures and version upgrades.

**Localization**: For supporting multiple theatres in different regions, incorporate localization at the API layer:

Support multiple languages and movie listings based on geography.

Handle different time zones for movie show timings.

**Data Transformation**: Implement transformation logic to convert legacy data formats to the platform's internal data format.

**3. Scaling to Multiple Cities, Countries, and Guaranteeing Platform Availability (99.99%)**

**Scaling Strategy**:

**Microservices Architecture**: Break down the platform into smaller, independent services (e.g., Booking, Payment, Notification), each of which can scale independently based on demand.

**Auto-scaling**: Use **container orchestration tools** like Kubernetes to scale services dynamically based on load.

**Global Distribution**: Deploy the system in multiple geographic regions using a multi-cloud approach (e.g., AWS, Azure, GCP) to serve users from nearby data centres and reduce latency.

**High Availability & Redundancy**:

Ensure **active-active failover** between different data centres.

Use **replication** and **sharding** for databases to ensure high availability and fault tolerance.

Set up **load balancers** for distributing traffic across multiple services and data centres.

**Monitoring and SLAs**:

Monitor system health using **Prometheus** and **Grafana** to track uptime, performance metrics, and potential issues.

Implement **circuit breakers** to prevent cascading failures in case of service disruptions.

**4. Integration with Payment Gateways**

**Design Approach**:

**Third-Party Payment Integration**: Use payment gateway APIs (e.g., Stripe, PayPal, Razorpay, etc.) for processing payments.

**Secure Payment Flow**: Ensure secure handling of payment information using **PCI-DSS** standards, encrypting sensitive data both at rest and in transit.

**Tokenization**: Use tokenization to securely store credit card details without saving them in the database.

**Retry Logic**: Implement retries and fallbacks for handling payment failures, ensuring that users are not left with a pending payment status.

**Webhooks**: Use webhooks from payment gateways to receive real-time notifications of successful payments, cancellations, and refunds.

**5. Monetizing the Platform**

**Revenue Models**:

**Ticket Sales**: Primary revenue from movie ticket bookings.

**Subscription Models**: Offer subscription plans for frequent users, with benefits like discounts and priority access.

**Theatre Partnerships**: Revenue share agreements with theatre partners, offering them a commission for using the platform.

**Advertisement**: Display targeted advertisements to users between movie bookings.

**Add-ons**: Offer premium services such as food and beverage ordering, VIP seats, and movie bundles.

**6. Protecting Against OWASP Top 10 Threats**

**Security Measures**:

**Injection**: Use parameterized queries and ORM frameworks (like Hibernate) to prevent SQL injection.

**Broken Authentication**: Use strong multi-factor authentication (MFA) for both customers and theatre partners.

**Sensitive Data Exposure**: Use **encryption** (AES-256) to protect sensitive data such as payment details and user information.

**XML External Entities (XXE)**: Disable external entities in XML parsers.

**Broken Access Control**: Implement **Role-Based Access Control (RBAC)** for restricting access to sensitive parts of the system.

**Security Misconfiguration**: Regularly update and patch systems, use secure default settings, and apply best practices in configuring web servers and applications.

**Cross-Site Scripting (XSS)**: Sanitize and validate all user inputs to prevent malicious scripts from executing.

**Insecure Deserialization**: Validate and securely deserialize objects before processing them.

**Using Components with Known Vulnerabilities**: Regularly update and patch all third-party libraries and components.

**Insufficient Logging & Monitoring**: Implement logging and monitoring solutions to detect and respond to potential security incidents quickly.

**7. Compliance Considerations**

**GDPR (General Data Protection Regulation)**: Ensure that personal data of users (e.g., emails, payment details) are handled according to GDPR guidelines, including obtaining user consent, data minimization, and providing users the right to access, rectify, and delete their data.

**PCI-DSS (Payment Card Industry Data Security Standard)**: Ensure compliance with PCI-DSS for securely handling and storing payment card data.

**SOC 2 Compliance**: For ensuring that security, availability, and confidentiality requirements are met for users and data.