Tab 1

[**1. Scalability 3**](#_y90f2kua9hxv)

[Requirement 3](#_2kdq0wnnawbw)

[Design Decisions 3](#_yi37djlsf2yn)

[Microservices Architecture 3](#_l0oc4li6ue63)

[Horizontal Scaling 3](#_8mmirgjko7xh)

[Event-Driven Architecture 4](#_t7yv8we8einz)

[Distributed Caching 4](#_19gcrqse1xfl)

[**2. Availability 4**](#_cajfg5t33qav)

[Requirement 4](#_ih8e9fefbkx1)

[Design Decisions 4](#_b45g7np5grzj)

[Replication 4](#_zcxl9pcqmfp1)

[Circuit Breaker Pattern 4](#_1v6kasku82tg)

[Load Balancing 4](#_g4xjbqcvkq52)

[Multi-Region Deployment 4](#_oao26n6knlj)

[**3. Performance 4**](#_3x8bq6aqkvo)

[Requirement 4](#_phm0sbb8wsgj)

[Design Decisions 5](#_25xwwap4kith)

[Database Optimization 5](#_v65hyun9yv5d)

[Concurrency Control 5](#_4xlgx0h85kox)

[Content Delivery Network (CDN) 5](#_vpvvu3yerksk)

[Service Profiling 5](#_wjyetrktjt3p)

[**4. Reliability 5**](#_tl4093rmpxig)

[Requirement 5](#_aho7beki8b8k)

[Design Decisions 5](#_6pqkz0mcuivp)

[Distributed Transactions 5](#_hmuli7nyetgz)

[Idempotent APIs 5](#_cm2aepfj87ms)

[Data Backup and Recovery 6](#_28hugdz9vghi)

[Chaos Engineering 6](#_tp55zb5cnief)

[**5. Security 6**](#_30i0wlwaruo)

[Requirement 6](#_dezas6synunr)

[Design Decisions 6](#_8vv8x4wq28q5)

[Authentication and Authorization 6](#_ubnszdffydta)

[Data Encryption 6](#_vuntvuj12dv)

[Rate Limiting and DDoS Protection 6](#_8rj9n85iiv8k)

[Audit Logging 6](#_4m2r3tva2d8i)

[**6. Maintainability 6**](#_vfbpb8gco7v2)

[Requirement 6](#_e3djg6fqu0q6)

[Design Decisions 7](#_tfnrkgr6w1v5)

[Service Discovery 7](#_drctvvi4bv6u)

[Centralized Logging 7](#_atsamhgv65k3)

[CI/CD Pipeline 7](#_40wt2n04ws4)

[**7. Observability 7**](#_kkryrngn1vph)

[Requirement 7](#_umc84vg7nomg)

[Design Decisions 7](#_v3lziwnwqbvs)

[Distributed Tracing 7](#_hbmad911i4r4)

[Health Checks 7](#_y1abyf697kyq)

[Alerting Systems 7](#_a9r6an7h2zqx)

[**8. Transactional Scenarios and Handling 7**](#_5ea01pvm766a)

[Seat Booking 8](#_5rwttktakpvg)

[Challenge 8](#_hd9tson3awo)

[Solution 8](#_x2efhelvtsir)

[Locking Mechanisms 8](#_9p6x4fx0a9oa)

[Retry Logic 8](#_k0uir2iwpdoo)

[Compensation Transactions 8](#_hk2aoona5w0p)

[Payment Processing 8](#_vmb1ac844tfm)

[Challenge 8](#_2hsybzkh0ndd)

[Solution 8](#_er64oj3hjk17)

[Two-Phase Commit (2PC) 8](#_r8kbmr8o19w8)

[Eventual Consistency 8](#_v075n52om3ta)

[Refund and Cancellation 9](#_4l8xcfpgjcve)

[Challenge 9](#_t9zrdn7hwb5g)

[Solution 9](#_3kt09t3zzc3a)

[Asynchronous Updates 9](#_r50uk0mabvx5)

[**9. OWASP 9**](#_s26c1890dlkh)

[Broken Access Control 9](#_sih9ijy0fqlg)

[Cryptographic Failures 9](#_nwhqr1fznjzr)

[Injection 9](#_8nwsf5fbj69y)

[Security Misconfiguration 9](#_cxq0oa9mvrd5)

[Vulnerable and Outdated Components 10](#_hej4dxmkelvt)

[Server-Side Request Forgery (SSRF) 10](#_t1npgnwapgqc)

[**10. Monetize Platform 10**](#_isahurgvdq95)

[Ticket Sales (Primary Revenue) 10](#_s8iiw1dr4n94)

[Service Fees 10](#_lzhkd9kbhyoh)

[Transaction Fees 10](#_afejx22ip3x1)

[Advertising 10](#_o0lu4nwgh6a2)

[Display Ads 10](#_r15mivdxvwp7)

[Premium Memberships or Subscriptions 10](#_4ljuyo26gdw7)

[VIP Membership 10](#_qz8metj1p3ck)

[Subscription Models 11](#_jdsv6jnk9j1n)

[Corporate Tie-ups and Bulk Booking 11](#_owy15zb7908)

[Corporate Sales 11](#_l7vzolk2xwg3)

[Group Discounts 11](#_swq300kj099i)

[Selling Merchandise 11](#_g42q7e8xty79)

[Affiliate Marketing 11](#_dktr0btit8ge)

[**11. Compliance 11**](#_1c5uz1q4w4k1)

[Data Privacy and Protection 11](#_my4k1w5b7vj8)

[Payment and Financial Compliance 11](#_buxidd7qh1tn)

[Consumer Protection Laws 12](#_1mx6vh2qd1q2)

[Tax Compliance 12](#_6i3mf5ak83b)

[Copyright and Intellectual Property 12](#_fmnjcghwekrt)

[Key Non-Functional Requirements and Design Solutions 1](#_5ic1cojp57mt)

# 1. Scalability

## Requirement

The system must handle high traffic during events (e.g., blockbuster movie releases).

## Design Decisions

### **Microservices Architecture**

* Each service (e.g., Theatre/Show/movie listings, ticket booking, payments, Notification, user) is independently deployable and scalable.

### **Horizontal Scaling**

* Create AWS ECS Cluster per Region.
* Use container orchestration platforms like **AWS ECS** to scale services dynamically based on traffic.
* Shard Database based on City id, and then shard by show Id,Booking Id inside a city.

### **Event-Driven Architecture**

* Use message brokers like **Kafka** to decouple services and process asynchronous requests during peak loads.

### **Distributed Caching**

* Use **Redis** to cache frequently accessed data (e.g., movie details, seating availability).

# 2. Availability

## Requirement

Ensure minimal downtime for seamless user experience.

## Design Decisions

### Replication

* Use database replication and service redundancy to avoid single points of failure.

### Circuit Breaker Pattern

* Implement tools like **Resilience 4j** to handle service failures gracefully.

### Load Balancing

* Use tools like **AWS ELB** to distribute traffic across multiple instances.

### Multi-Region Deployment

* Deploy the system across multiple regions for geo-redundancy using cloud platforms like **AWS**.

# 3. Performance

## Requirement

Low latency for transactional operations like seat booking.

## Design Decisions

### Database Optimization

* Use optimized queries, indexing,views,stored procedures and read replicas for high-frequency reads.

### Concurrency Control

* Use pessimistic locking mechanisms to handle simultaneous seat bookings.

### Content Delivery Network (CDN)

* Use a CDN for serving static content like images and movie trailers.

### Service Profiling

* observability tools (Amazon Cloud watch ) to identify metrics to monitor, CPUUtilization, Percentage of CPU used,MemoryUtilization, Percentage of memory used. set alarm and optimize bottlenecks.

# 4. Reliability

## Requirement

Ensure data consistency and fault tolerance during transactions.

## Design Decisions

### Distributed Transactions

* Implement the **Saga pattern** for handling distributed transactions across microservices.

### Idempotent APIs

* Ensure API endpoints can handle retries without unintended side effects.

### Data Backup and Recovery

* Schedule periodic backups and maintain versioned snapshots of critical data.

### Chaos Engineering

* Use tools like **Gremlin** to simulate failures and test system reliability under stress and **Jmeter** for load test.

# 5. Security

## Requirement

Protect user data and transactions.

## Design Decisions

### Authentication and Authorization

* Use OAuth 2.0 with a token-based system (e.g., **JWT**) for secure access.

### Data Encryption

* Encrypt sensitive data in transit (TLS) and at rest (AES-256).

### Rate Limiting and DDoS Protection

* Use AWS API Gateway and rate-limiting algorithms like the **token bucket**.

### Audit Logging

* Maintain detailed logs for sensitive transactions and monitor with tools like **ELK Stack**.

# 6. Maintainability

## Requirement

Ensure ease of updates and debugging.

## Design Decisions

### Service Discovery

* Use service registries like **ECS Managed Service discovery** for managing microservice endpoints.

### Centralized Logging

* Use logging frameworks (e.g.**Logstash**) to consolidate logs.

### CI/CD Pipeline

* Automate testing, deployment, and rollback using tools like **Jenkins** or **GitHub Actions**.

# 7. Observability

## Requirement

Monitor system health and performance in real-time.

## Design Decisions

### Distributed Tracing

* Use tools like **Spring cloud Sleuth** to add common message id across micro services and unique id for specific micro services to trace requests across services.

### Health Checks

* Implement health endpoints in services for proactive monitoring by the orchestrator.

### Alerting Systems

* Integrate SNS with the AWS service generating alerts, such as CloudWatch Alarms Configure alerts in **Email/Slack** for anomalies detected by monitoring tools.

# 8. Transactional Scenarios and Handling

Here’s how design solutions align with specific transactional scenarios:

## Seat Booking

### **Challenge**

Concurrent users might try to book the same seat.

### **Solution**

#### **Locking Mechanisms**

* Use PostgresQL advanced locking mechanisms (e.g., advisory locks) and better concurrency handling via MVCC (Multiversion Concurrency Control). to avoid double booking.

#### **Retry Logic**

* Implement retry with exponential backoff for failed transactions.

#### **Compensation Transactions**

* Allow rollbacks for unsuccessful bookings using the Saga pattern.

## Payment Processing

### Challenge

Ensuring consistency in a multi-step process involving third-party systems.

### Solution

#### Two-Phase Commit (2PC)

* For tightly coupled systems, but avoid overuse due to performance impact.

#### Eventual Consistency

* Use Message Queue to reconcile transactions asynchronously.

## Refund and Cancellation

### Challenge

Ensure quick refunds while reconciling with external payment gateways.

### Solution

#### Asynchronous Updates

* Notify users and payment gateways via message queues.
* Payment failed/Cancelled kafka queue for Refund processing.

# 9. OWASP

## Broken Access Control

* Use Spring Security to enforce access control rules.
* Implement role-based or attribute-based access control policies.
* Validate @PreAuthorize and @PostAuthorize annotations to secure methods.

## Cryptographic Failures

* Use strong algorithms like AES-256, RSA-2048, and PBKDF2 for encryption and hashing.
* Always store passwords using BCryptPasswordEncoder in Spring Security.
* Enforce HTTPS to secure data in transit

## Injection

* Use Spring Data JPA or Hibernate, which provides protection through parameterized queries.
* Sanitize and validate user inputs rigorously.
* Avoid dynamic query building using string concatenation; use @Query with placeholders in Spring Data JPA.
* Use libraries like Hibernate Validator to validate input data.

## Security Misconfiguration

* Disable directory listing, stack traces, and verbose error messages in production.
* Use @ConfigurationProperties to securely bind configurations.
* Regularly review and harden security settings, such as CORS and CSRF.

## Vulnerable and Outdated Components

* Use dependency management tools like Maven or Gradle to track versions.
* Remove unused dependencies.

## Server-Side Request Forgery (SSRF)

* Validate and whitelist URLs in any server-to-server HTTP requests.
* Disable access to local or private networks from your server.
* Implement rate-limiting and authentication for APIs handling such requests.

# 10. Monetize Platform

## Ticket Sales (Primary Revenue)

### Service Fees

* The platform charges a service fee or convenience fee on each ticket sold. This can be a fixed amount or a percentage of the ticket price.

### Transaction Fees

* Platform can also earn a small fee for processing payments, either as a percentage or a flat rate.

## Advertising

### Display Ads

* The platform can run display advertising on the website or mobile app. These ads could be for upcoming events, movies, or other entertainment-related products.

## Premium Memberships or Subscriptions

### VIP Membership

* Platform can offer a paid membership program, giving users perks like early access to tickets, discounts, or exclusive events.

### Subscription Models

* Monthly or yearly subscription plans for frequent users offering benefits like discounted tickets, priority booking, etc.

## Corporate Tie-ups and Bulk Booking

### Corporate Sales

* Platform can collaborate with businesses for bulk booking of tickets for corporate events, employee engagement, or incentive programs.

### Group Discounts

* Offer group discounts or corporate packages for large bookings, often generating higher volumes of sales.

## Selling Merchandise

### Affiliate Marketing

* Partnering with merchandise sellers and earning a commission on items sold through affiliate links.

# 11. Compliance

## Data Privacy and Protection

* General Data Protection Regulation (GDPR) in the EU.
* California Consumer Privacy Act (CCPA) in the US.
* Personal Data Protection Bill (PDPB) or Digital Personal Data Protection Act (DPDP Act) in India.

## Payment and Financial Compliance

* Payment Card Industry Data Security Standard (PCI DSS): For handling credit/debit card transactions securely.
* Reserve Bank of India (RBI) guidelines for digital payment systems.

## Consumer Protection Laws

* Clearly state refund, cancellation, and rescheduling policies.
* Display accurate event information (pricing, location, time)

## Tax Compliance

* Collect and remit applicable taxes, such as Goods and Services Tax (GST) in India or VAT in the EU, for ticket sales

## Copyright and Intellectual Property

* Obtain necessary rights for event details, images, or promotional material displayed on the platform.
* Avoid hosting or distributing copyrighted content without permission.

By aligning these solutions with NFRs, the backend becomes robust, scalable, and user-friendly.