Movie Ticket Booking Platform (MTBP) — Architecture & Design Document

Document Control

- **Title:** Movie Ticket Booking Platform (MTBP) — Architecture & Design
- **Version:** 1.0
- **Author:** Sonil Kumar
- **Date:** 2025-10-02
- **Status:** Review
- **Distribution List:** Theatre Owners & Movie Viewers
- **Revision History**
- v1.0 — Initial template — 2025-10-02 — Sonil

Table of Contents

8. Platform Provisioning & Hosting

- 8.1 Cloud/Hybrid strategy
- 8.2 Region, AZ, and multi-GEO considerations
- 8.3 Sizing & capacity planning
- 8.4 Infrastructure as Code (IaC)

9. Data Management & Transactions

- 9.1 Data modelling (ER / high-level schema)
- 9.2 Transactional guarantees & patterns
- 9.3 Eventual consistency and compensating transactions
- 9.4 Backups, retention & archival

10. COTS & Enterprise Systems

11. CI/CD, Release Management & Rollout Strategy

12. Scalability, Availability & Resilience

13. Performance & Capacity Planning

14. Security, Compliance & Privacy

15. Operational Readiness & Runbooks

16. Cost Estimates & Licensing

17. Risk Assessment & Mitigation

18. Stakeholder Management & Decision Logs

19. Appendices

- A. Glossary
- B. Contact matrix
- C. Important Links

1. Executive Summary

The **Movie Ticket Booking Platform (MTBP)** is a highly scalable, resilient, and globally deployable platform designed to enable end customers to search, browse, and book movie tickets online. The platform caters to multiple stakeholders including:

- **End Customers:** Moviegoers who search movies, select theatres, and book seats.
- **Theatre Partners:** Operators providing show schedules, seat inventory, and venue details via APIs or portal interfaces.
- **MTBP Admins:** Platform operators managing theatre onboarding, promotions, and analytics.

Key Capabilities:

1. **Real-time Seat Booking:** Allows customers to view live seat availability and confirm bookings without conflicts, supporting high-concurrency scenarios.
2. **Partner Integrations:** Seamlessly integrates with existing theatre IT systems and supports onboarding new partners with a flexible API/portal.
3. **Payment Gateway Integration:** Ensures secure and compliant payments using multiple third-party providers.
4. **Localization & Geo-scaling:** Supports multi-language, multi-currency, and city-level regional content for international expansion.
5. **Observability & Metrics:** Comprehensive monitoring with real-time dashboards (Grafana), alerting, and logging (Prometheus & Loki).
6. **Event-driven Architecture:** Uses Kafka for asynchronous inter-service communication, enabling scalability, loose coupling, and resilience.
7. **High Availability & Resilience:** Designed for **99.99% uptime**, with automated failover, multi-region deployment, and disaster recovery.

Business Benefits:

- Streamlined booking experience increases customer engagement and conversion rates.
- Flexible partner onboarding reduces integration costs and time-to-market.
- Scalable infrastructure supports rapid expansion into new cities and countries.
- Robust security and compliance architecture minimizes operational risk.

2. Scope & Objectives

In Scope

The MTBP project will deliver:

1. **Core Booking System**
    - Movie search by location, theatre, date, or genre.
    - Real-time seat selection and reservation.
    - Ticket confirmation and refund workflows.

2. **Partner Integration Framework**
    - APIs for existing theatre systems.
    - Web portal for manual data entry for new theatre partners.
    - Automated reconciliation of theatre inventory and show schedules.

3. **Customer Management**
    - Registration, profile management, loyalty, and preferences.

4. **Payment Processing**
    - Integration with multiple third-party payment gateways (PCI-DSS compliant).
    - Support for multi-currency and local payment methods.

5. **Admin & Reporting**
    - Analytics dashboards for bookings, revenue, cancellations, and theatre utilization.
    - Partner performance and audit reporting.

6. **Observability**
    - Metrics collection, logging, tracing, and alerting.
    - Dashboards for business and operational monitoring.

Out of Scope

- Third-party analytics integrations not directly required for booking operations.
- Offline theatre-only ticketing or kiosk-based management (unless explicitly onboarded).

Objectives

- Deliver a **secure, resilient, and highly available** booking platform capable of scaling horizontally.
- Enable **fast onboarding of new theatres** and movie content.
- Support **internationalization** with localization of languages, currencies, and content.
- Provide **observability, monitoring, and alerting** for operational excellence.
- Ensure **regulatory compliance** (PCI-DSS, GDPR, local payment laws) across geographies.
- Facilitate a **robust CI/CD pipeline** and blue/green deployment strategy for seamless releases.

3. Stakeholders & Roles

The success of the **Movie Ticket Booking Platform (MTBP)** depends on clear roles, responsibilities, and active engagement from all stakeholders. Below is a detailed breakdown of key stakeholders:

3.1 Product Owner

- **Role:** Represents the business and end-customer interests.
- **Responsibilities:**
    - Define product vision, roadmap, and feature priorities.
    - Approve functional requirements and user stories.
    - Engage with marketing, theatre partners, and finance teams for alignment.
- **Decision Authority:** Final sign-off on product features, business requirements, and release priorities.

3.2 Enterprise Architect

- **Role:** Oversees enterprise-wide technology alignment and architectural compliance.
- **Responsibilities:**
    - Define architectural principles, design patterns, and non-functional requirements.
    - Ensure platform scalability, high availability, and security compliance.
    - Review technical decisions from Solution Architects.
- **Decision Authority:** Approves high-level technology choices and enterprise integration patterns.

3.3 Solution Architect

- **Role:** Designs end-to-end system solutions and ensures implementation aligns with architecture.
- **Responsibilities:**
    - Design service decomposition, data flows, and integration patterns.
    - Approve API contracts, microservice boundaries, and CI/CD pipelines.
    - Review technical design documents submitted by engineering teams.
- **Decision Authority:** Sign-off on solution-level architecture and design decisions.

3.4 Engineering Leads

- **Roles:** Lead respective engineering disciplines (Frontend, Backend, Data, Platform).
- **Responsibilities:**
    - Ensure implementation follows coding standards and architecture guidelines.
    - Plan and deliver sprint work aligned with the product roadmap.
    - Conduct peer code reviews and ensure high-quality code.
- **Decision Authority:** Technical decisions within their domain (e.g., tech stack, libraries, frameworks).

3.5 QA & SRE (Site Reliability Engineering)

- **Role:** Ensure platform quality, reliability, and operational readiness.
- **Responsibilities:**
    - Develop test plans, automation suites, and conduct performance testing.
    - Monitor system availability, error rates, and SLO/SLA adherence.
    - Implement alerting, incident response, and disaster recovery procedures.
- **Decision Authority:** Can halt release if critical issues are found; approve operational readiness.

3.6 Security & Compliance

- **Role:** Ensure adherence to security standards and regulatory requirements.
- **Responsibilities:**
    - Review and approve secure authentication/authorization mechanisms.
    - Conduct vulnerability assessments, penetration testing, and audits.
    - Ensure PCI-DSS, GDPR, and regional payment compliance.
- **Decision Authority:** Final approval for production readiness from a security perspective.

3.7 Theatre Partnership Managers

- **Role:** Manage relationships with theatre owners and content providers.
- **Responsibilities:**
    - Onboard new theatre partners and coordinate data integration.
    - Ensure show schedules, pricing, and inventory data are accurate.
    - Monitor partner performance metrics and feedback.
- **Decision Authority:** Authorize theatre onboarding and data integration approvals.

3.8 Payment Gateway / Vendor Teams

- **Role:** Provide payment processing services and integration support.
- **Responsibilities:**
    - Ensure secure transaction processing and tokenization compliance.
    - Support integration of multiple payment methods, currencies, and refunds.
    - Provide operational metrics, SLAs, and reconciliation reports.
- **Decision Authority:** Approve integration approach and transaction settlement rules.

3.9 Regional Operations Teams

- **Role:** Manage platform operations across geographies.
- **Responsibilities:**
    - Ensure platform availability and performance in each region.
    - Coordinate deployments, maintenance, and localized content updates.
    - Handle local support, incident response, and compliance reporting.
- **Decision Authority:** Local operational decisions affecting uptime, scaling, and regional content.

4. Business Requirements (Functional)

These below requirements, the platform must do to meet the needs of **end customers, theatre partners, and administrators**. Grouped into logical domains for clarity.

4.1 Customer-Facing Features

4.1.1 Movie Discovery

- **Description:** Customers can search and browse movies by city, theatre, genre, language, date, and showtime.
- **Key Capabilities:**
    - Location-based search with map view.
    - Filter by genre, language, popularity, ratings.
    - Personalized recommendations based on viewing history.
- **User Story:** *As a moviegoer, I want to find available movies and showtimes quickly so that I can plan my visit efficiently.*

4.1.2 Seat Selection & Booking

- **Description:** Real-time seat availability and booking confirmation.
- **Key Capabilities:**
    - Display live seat map for each theatre and show.
    - Allow customers to select multiple seats in one booking.
    - Prevent double-booking using **inventory locking** (via API or Kafka events).
    - Support refunds, cancellations, and partial booking modifications.
- **User Story:** *As a customer, I want to select my preferred seats and book them instantly to avoid conflicts.*

4.1.3 Payment Processing

- **Description:** Secure, multi-channel payments integrated with third-party gateways.
- **Key Capabilities:**
    - Support multiple payment methods: credit/debit cards, wallets, UPI, local payment methods.
    - Multi-currency support for international customers.
    - Generate receipts, confirmations, and transaction logs.
- **User Story:** *As a customer, I want to pay securely and receive instant confirmation of my ticket purchase.*

4.1.4 Customer Account & Loyalty

- **Description:** Profile management, loyalty points, and preferences.
- **Key Capabilities:**
    - Register/login via email, mobile, or OAuth2 providers (e.g., Google, Facebook).
    - View booking history, upcoming shows, and loyalty points.
    - Save preferences for preferred theatres or genres.
- **User Story:** *As a customer, I want to manage my profile and loyalty points to get personalized offers.*

4.2 Theatre Partner Features

4.2.1 Partner Onboarding

- **Description:** Onboard both legacy and new theatres via portal/API.
- **Key Capabilities:**
    - Upload theatre metadata, show schedules, pricing, and seat layouts.
    - Validate data consistency and completeness.
- **User Story:** *As a theatre partner, I want to quickly register and update theatre information to start selling tickets.*

4.2.2 Inventory Management

- **Description:** Manage seat availability and pricing dynamically.
- **Key Capabilities:**
    - Real-time seat availability updates.
    - Automated synchronization with MTBP inventory service.
- **User Story:** *As a theatre partner, I want to ensure my inventory is accurately reflected on the platform.*

4.3 Admin Features

4.3.1 Operational Management

- **Description:** Central administration for platform operations.
- **Key Capabilities:**
    - Manage users, theatre partners, and show schedules.
    - Monitor platform performance, bookings, payments, and disputes.
    - Generate reports and analytics for revenue, utilization, and partner performance.
- **User Story:** *As an admin, I want to monitor and manage all platform operations to ensure smooth business functioning.*

4.3.2 Pricing, Promotions & Refunds

- **Description:** Admin can manage discounts, promotions, and handle refunds.
- **Key Capabilities:**
    - Apply region-specific or theatre-specific promotions.
    - Define automated refund rules based on cancellations.
    - Track promotional effectiveness via dashboards.
- **User Story:** *As an admin, I want to create promotions and manage refunds to enhance customer satisfaction and drive sales.*

4.4 Integration Features

4.4.1 Partner System Integration

- **Description:** Interface with theatre legacy systems.
- **Key Capabilities:**
    - REST APIs or batch data ingestion for show schedules and inventory.
    - Event-driven updates using Kafka for inventory or booking events.
- **User Story:** *As a system, I want to synchronize theatre inventory and bookings in real-time to prevent inconsistencies.*

4.4.2 Payment Gateway Integration

- **Description:** Multi-gateway support with fallback.
- **Key Capabilities:**
    - Automatic retry and reconciliation for failed payments.
    - Multi-region support and failover to alternate gateways.
- **User Story:** *As a system, I want to process payments reliably to maintain revenue and customer trust.*

4.5 Priority Matrix

Feature Domain	Priority	Rationale
Seat Selection & Booking	High	Core business capability
Payment Processing	High	Critical for revenue and trust
Movie Discovery	Medium	Enhances customer experience
Partner Onboarding	High	Essential for scaling platform coverage
Customer Account & Loyalty	Medium	Increases retention and engagement
Admin Dashboards	Medium	Required for operations oversight
Promotions & Refunds	Medium	Business flexibility and customer satisfaction
System Integrations	High	Ensures real-time inventory & payment accuracy

Functional Requirement Traceability Table (FRTT)

#	Functional Requirement	Responsible Service / Component	API / Integration Point	Primary Stakeholder
1	Search movies by city & theatre	Customer Service	`/movies/search`	End Customer
2	Showtimes & seat-map visualization	Inventory Service / Theatre Service	`/inventory/showtimes`, `/theatre/shows`	End Customer, Theatre Partner
3	Real-time seat availability & reservations	Booking Service, Inventory Service	`/booking/reserve`, Kafka Events `InventoryUpdated`	End Customer, Theatre Partner
4	Multi-currency pricing, promotions & refunds	Payment Service, Admin Service	`/payment/charge`, `/promotions/apply`, `/payment/refund`	End Customer, Admin
5	Partner onboarding portal & APIs	Theatre Partner Service	`/partner/onboard`, `/partner/update`	Theatre Partner, Admin
6	Admin dashboards & reporting	Admin Service, Grafana dashboards	Internal REST APIs, Grafana	Admin, Product Owner
7	Audit trails & payment reconciliation	Payment Service, Booking Service	`/payment/transactions`, Kafka Events `BookingCreated/Cancelled`	Admin, Finance Team
8	Integration with payment gateways	Payment Service	External Payment Gateway APIs	Payment Gateway Teams
9	Partner system integration	Theatre Partner Service	REST / SOAP / Batch APIs	Theatre Partner Managers
10	Loyalty and profile management	Customer Service	`/customer/profile`, `/customer/loyalty`	End Customer

5. Non-Functional Requirements (NFRs)

These are measurable, enforceable, and critical for platform stability, scalability, and compliance.

5.1 Availability

- **Target SLA:** 99.99% uptime per region.
- **Implementation Considerations:**
  - Multi-AZ deployment per region to handle single-zone failures.
  - Multi-region active-active failover for disaster recovery.
  - Automated health checks, load balancing, and traffic rerouting.
  - Redundant service instances for all microservices and supporting databases.
- **Rationale:** Ensures high reliability for end customers and theatre partners, minimizing booking conflicts and downtime.

5.2 Performance

- **Target Metrics:**
  - 95th percentile response time for search queries: < 300ms.
  - End-to-end booking flow (seat selection → confirmation → payment): < 1.5s under normal load.
  - Payment processing and reconciliation: < 2s under peak load.
- **Implementation Considerations:**
  - Caching layer for frequently queried data (e.g., movies, theatre info).
  - Asynchronous processing for non-critical workflows (Kafka for event-driven updates).
  - Database indexing, read replicas, and query optimization.
- **Rationale:** Enhances customer experience, reduces abandonment, and supports high-volume transactions during peak hours.

5.3 Scalability

- **Horizontal Scalability:** All services should scale independently based on load.
- **Peak Load Targets:** Support up to ~50,000–100,000 **peak** (RPS) without degradation.
- **Implementation Considerations:**
  - Containerized microservices with auto-scaling policies.
  - Event-driven architecture using Kafka to decouple service dependencies.
  - Partitioned databases or sharding to distribute load.
- **Rationale:** Supports expansion to multiple cities/countries and high traffic events (e.g., blockbuster releases).

5.4 Security

- **Authentication & Authorization:**
    - OAuth2 / OpenID Connect for secure login (end customers, admins, partners).
    - Role-based access control (RBAC) for API and admin portal access.
- **Data Protection:**
    - PCI-DSS compliance for payment card data.
    - Encryption of sensitive data at rest (AES-256) and in transit (TLS 1.2+).
- **Operational Security:**
    - Secure API endpoints, input validation, and logging of access attempts.
    - Periodic security audits and penetration testing.
- **Rationale:** Ensures customer trust, regulatory compliance, and protection of platform integrity.

5.5 Localization

- **Languages:** Support multiple languages including English, local languages per country.
- **Currency:** Support multiple currencies and regional payment methods.
- **Locale-specific Content:** Date, time, and number formatting according to region.
- **Implementation Considerations:**
    - Configuration-driven internationalization (i18n) in UI and backend services.
    - Locale-based content delivery using microservice configuration or CDN.
- **Rationale:** Enhances usability, customer adoption, and international expansion readiness.

5.6 Observability

- **Metrics:** Collect metrics at service-level (Prometheus) and business-level (number of bookings, revenue, cancellations).
- **Logging:** Centralized logging (Loki) with structured log formats for auditing and debugging.
- **Tracing:** End-to-end distributed tracing to track booking and payment workflows.
- **Alerting:** Configurable alerts for SLA breaches, errors, or abnormal patterns.
- **Runbooks:** Documented operational procedures for common incidents.
- **Rationale:** Enables proactive monitoring, incident resolution, and platform reliability.

5.7 Reliability & Resilience

- **Circuit Breakers:** Prevent cascading failures in microservices.
- **Retries with Exponential Backoff:** For transient failures in downstream services.
- **Failover:** Automatic failover for critical services (DB replicas, multi-region services).
- **Data Consistency:** Eventual consistency using asynchronous messaging, compensating transactions for critical workflows.
- **Rationale:** Ensures platform stability under partial failures and high load scenarios.

Summary of Rational Numbers for MTBP NFRs:

Parameter	Recommended Value
Peak Load (RPS)	100,000 RPS globally
Average Load (RPS)	5,000 RPS
Search Latency	< 300 ms (95th percentile)
Booking Flow Latency	< 1.5 s
Payment Processing Latency	< 2 s
SLA / Availability	99.99% uptime
Scaling	Horizontal autoscaling for all microservices
Multi-region Deployment	Yes, active-active with failover

6. Architecture Overview

We have created a C4 Design repository in Github to maintain the diagrams maintenance.
GitHub Repository Link:
https://github.com/skynovicecoder/mtbp-c4-diagrams

6.1 Context Diagram (C4 — Level 1)

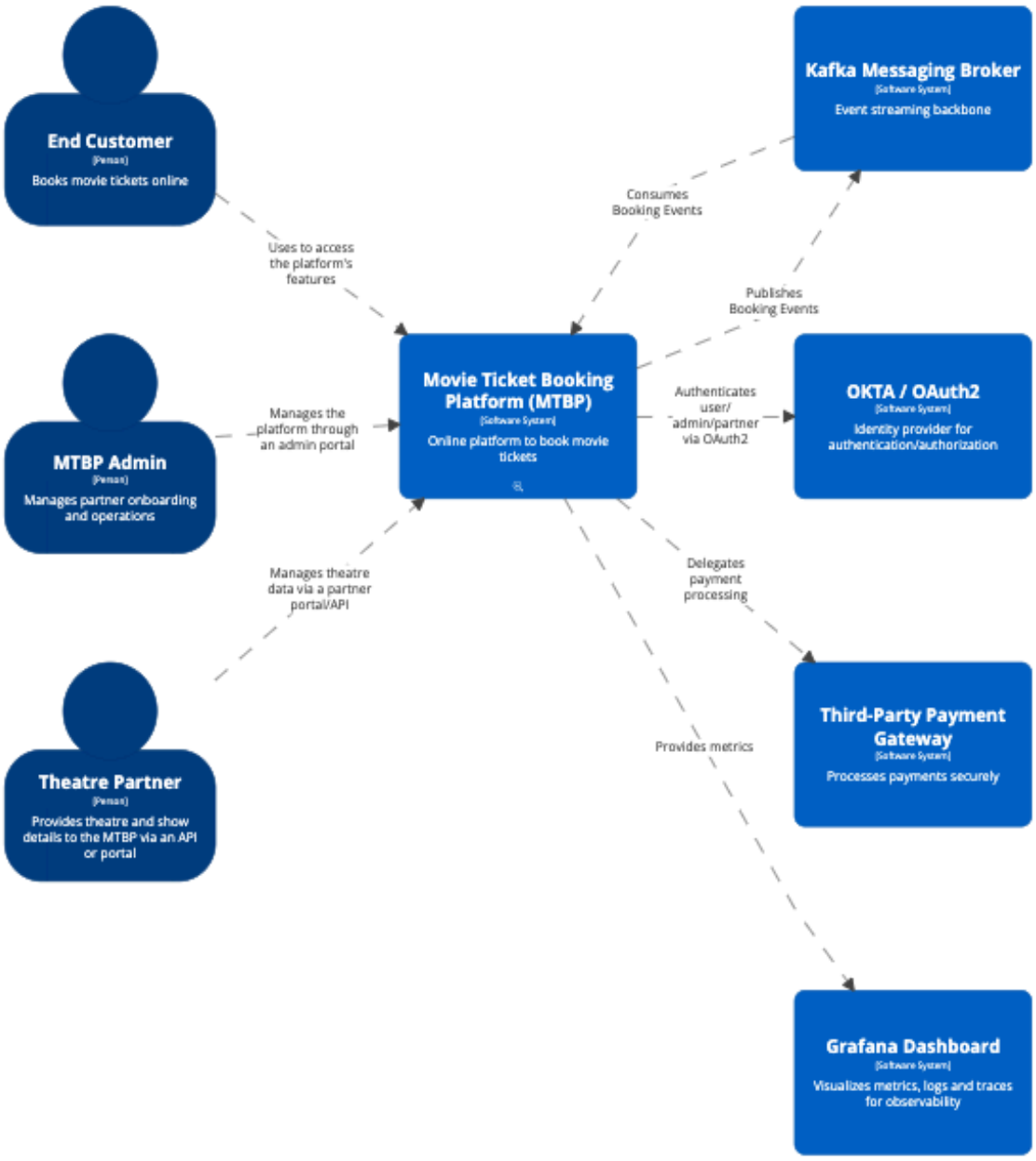
Purpose: Shows the **MTBP system in its environment**, highlighting interactions with users, partners, and external systems.

Key Entities:

- **Users:** End customers (moviegoers) interacting with web or mobile apps.
- **Theatre Partners:** External systems providing show schedules and seat inventory.
- **MTBP Admins:** Internal operators managing the platform.
- **External Systems:**
    - Payment Gateways (PCI-DSS compliant)
    - Identity Provider (OKTA / OAuth2)
    - Observability tools (Grafana, Prometheus, Loki)

Interactions:

- Users access the MTBP via web/mobile apps → routed through Load Balancer → API Gateway → microservices.
- Theatre partners integrate via REST APIs or portals.
- Payment services interact with external payment gateways.
- Observability systems collect metrics and logs from all microservices.



[System Context] Movie Ticket Booking Platform (MTBP)

Thursday 2 October 2025 at 23:30 India Standard Time

6.2 Container Diagram (C4 — Level 2)

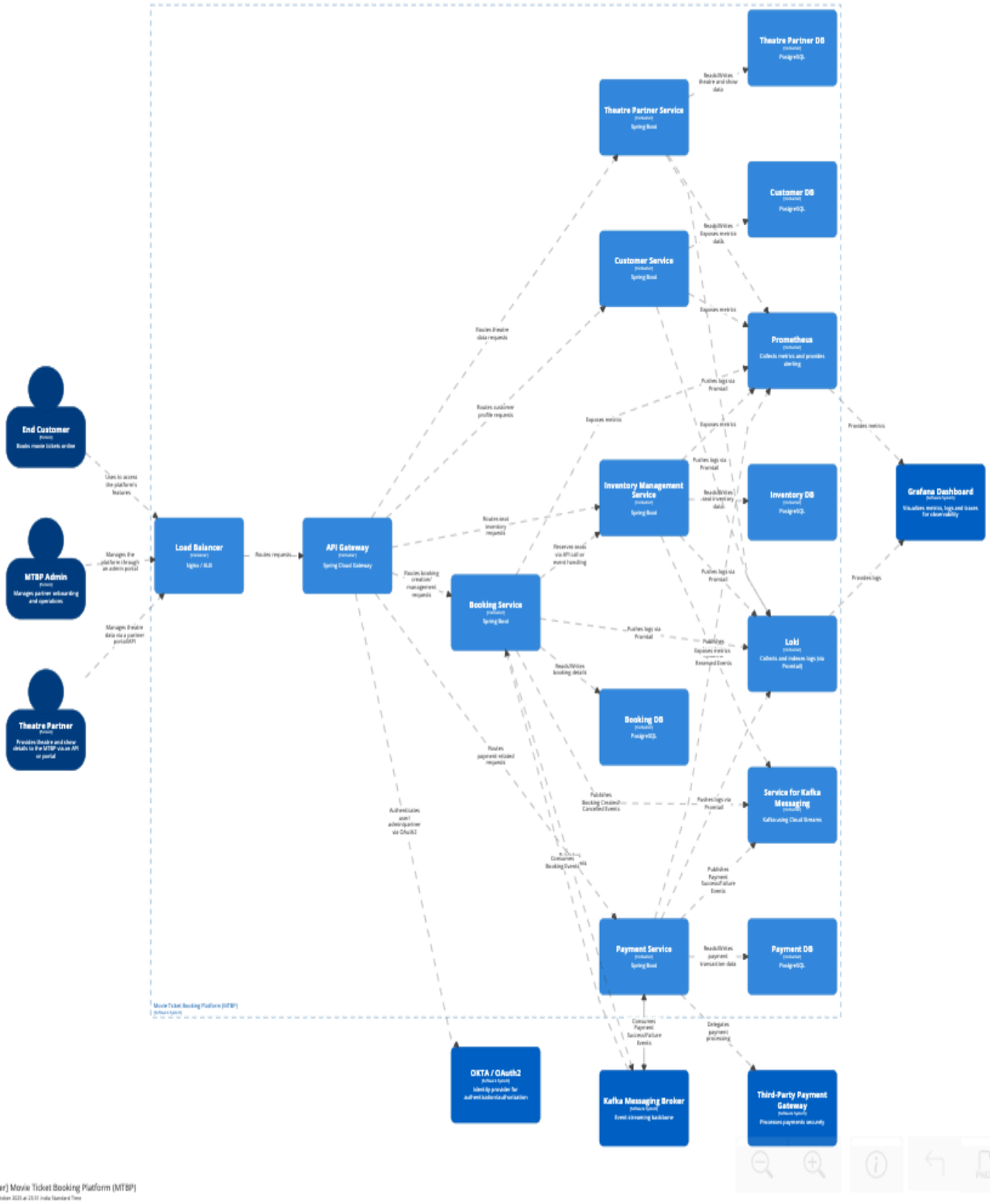
Purpose: Shows the **internal containers** of MTBP — how major services and databases are structured.

Key Containers:

- **Load Balancer (Nginx / ALB)**: Handles incoming traffic and distributes requests.
- **API Gateway (Spring Cloud Gateway)**: Routes requests to microservices, performs authentication via OKTA.
- **Microservices (Spring Boot):**
    - **Customer Service:** Manages user profiles and loyalty.
    - **Theatre Partner Service:** Manages theatre data and partner onboarding.
    - **Inventory Service:** Manages seat availability and inventory events.
    - **Booking Service:** Handles booking creation, cancellation, and events.
    - **Payment Service:** Processes payments, refunds, and reconciliation.
- **Databases (PostgreSQL):**
    - Customer DB, Theatre DB, Inventory DB, Booking DB, Payment DB.
- **Messaging Layer (Kafka):**
    - Asynchronous communication for events like booking created, inventory reserved, payment success/failure.
- **Observability:**
    - Prometheus (metrics), Loki (logs), Grafana (dashboards).

Interactions:

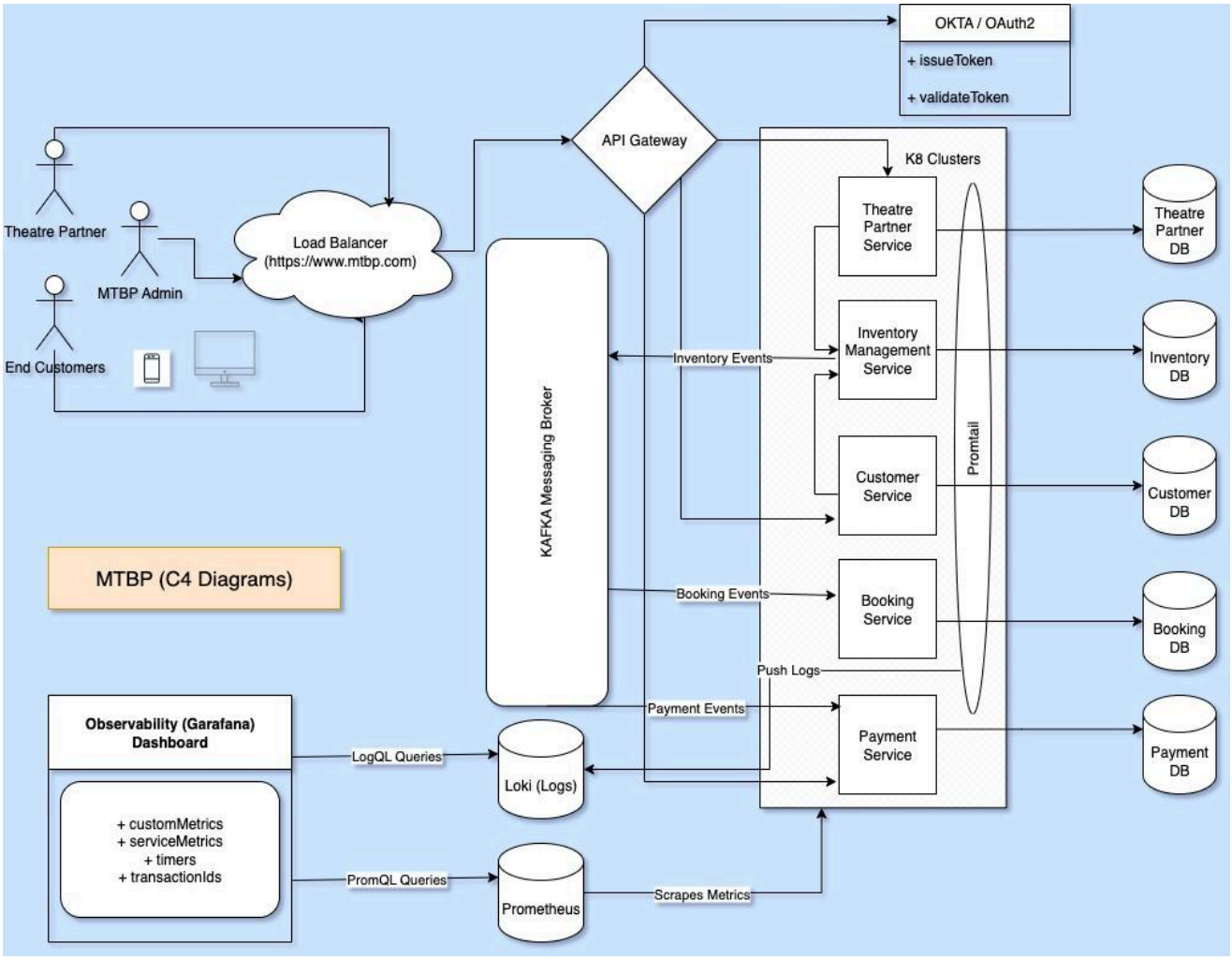
- Users/admins → LB → API Gateway → respective microservices → DB or Kafka.
- Services publish/subscribe to Kafka for decoupled event-driven communication.
- Metrics/logs pushed to Prometheus/Loki → Grafana dashboards.



(Container) Movie Ticket Booking Platform (MTBP)

Thursday 2 October 2025 at 23:51 India Standard Time

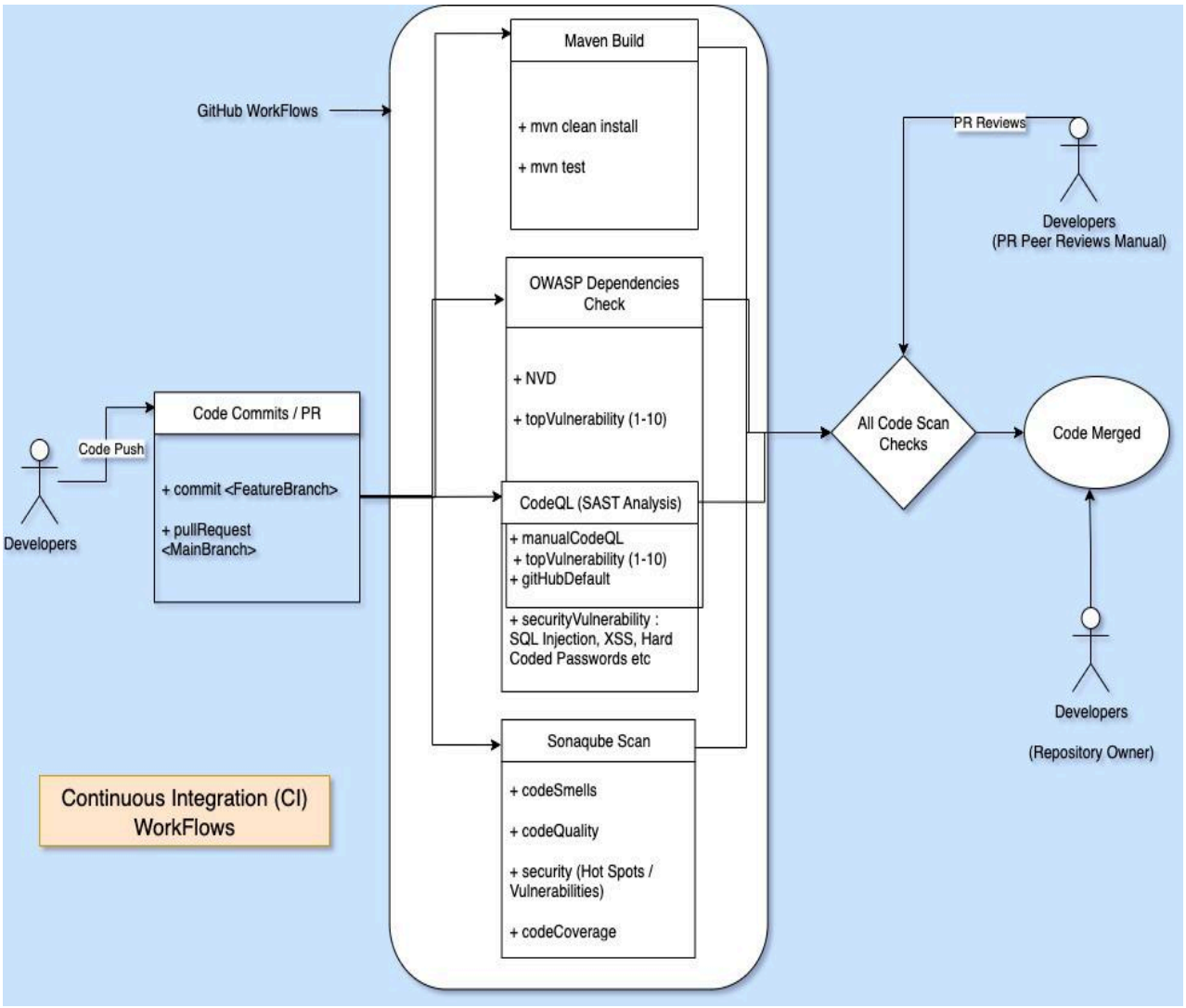
6.3 UML Diagrams For Full Flow Understanding of MTBP



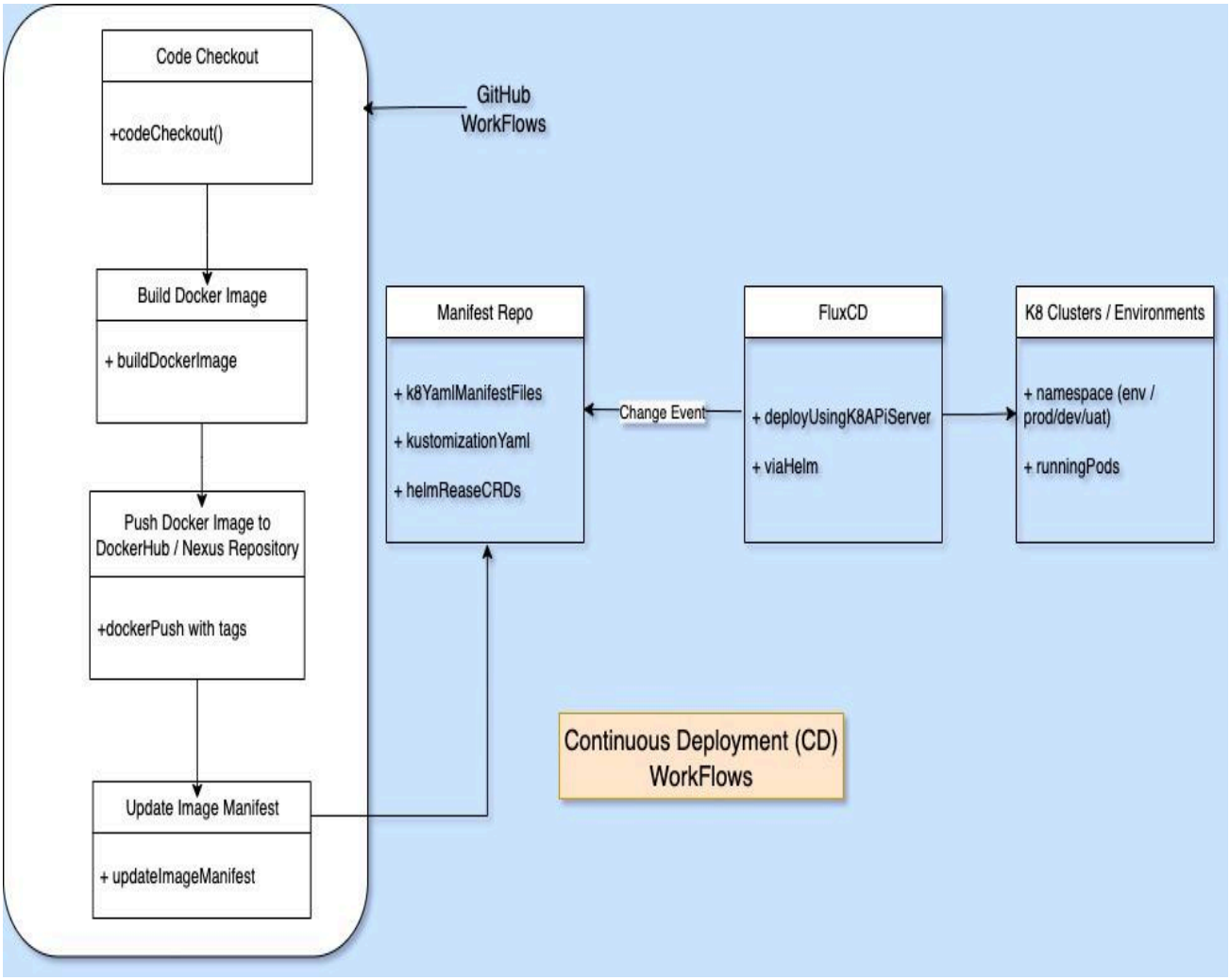
6.4 Deployment Diagrams (CI / CD)

Multi-region deployment: Active-active setup per region.
Service containers running in Kubernetes with autoscaling.
Kafka cluster for event streaming (multi-broker).
Observability stack: Prometheus, Loki, Grafana with alerting.
CI/CD integration: GitHub Actions / Jenkins pipelines with canary or blue-green deployments.

CI (Continuous Integration) Workflow



CD (Continuous Deployments) Workflow



7. Detailed Design

This section details how each MTBP microservice is implemented, integrated, and monitored, including data flows, API definitions, and architectural patterns.

7.1 API Gateway & Edge Architecture

The **API Gateway & Edge layer** acts as the **front-door** to the MTBP ecosystem, providing secure, scalable, and controlled access to backend microservices. It ensures **consistent cross-cutting concerns** such as authentication, authorization, rate-limiting, observability, and request routing.

Objectives of the API Gateway Layer

1. **Unified Entry Point** – A single domain and endpoint for external clients (customers, theatres, admins, partners).
2. **Security Enforcement** – Enforce OAuth2/OpenID Connect tokens via OKTA, validate JWTs, manage roles and permissions.
3. **Traffic Management** – Apply throttling, rate limiting, and request shaping to protect backend services.
4. **Routing & Composition** – Route to correct microservice or aggregate multiple services for composite responses.
5. **Resiliency & Fault Tolerance** – Apply retries, circuit breakers, and caching where applicable.
6. **Observability Hooks** – Request tracing, logging, and metrics collection for monitoring.
7. **Geo-aware Routing** – Direct requests to the nearest region (multi-geo deployments).

Architecture Components

Component	Responsibility
Load Balancer (Nginx / ALB)	First edge entry point; distributes traffic across API Gateway nodes.
API Gateway (Spring Cloud Gateway / Kong / Apigee)	Enforces policies (auth, rate limiting, logging, request transformation) and routes to microservices.

AuthN/AuthZ via OKTA	Validates OAuth2/OpenID Connect tokens; integrates with RBAC policies.
WAF (Web Application Firewall)	Protects against common attacks (SQL injection, XSS, DDoS).
Edge Caching/CDN	Static assets (UI, images, posters) cached at edge locations for performance.
Service Registry (Eureka/Consul)	Dynamic discovery of backend services for routing.
Circuit Breaker Layer (Resilience4j)	Handles degraded dependencies gracefully.

Request Flow

1. **Client Initiates Request**
   End-customer or partner sends HTTPS requests (e.g., booking tickets, fetching showtimes).
2. **Load Balancer Receives Traffic**
   Balancer (e.g., AWS ALB, Nginx ingress) distributes incoming requests across healthy API Gateway instances.
3. **API Gateway Processing**
   - **Authentication/Authorization:** Validates access token via OKTA.
   - **Rate Limiting:** Applies per-user, per-service, or per-geo quotas.
   - **Request Transformation:** Converts external API schema to internal service schema if needed.
   - **Routing:** Forwards request to appropriate microservice (e.g., `Booking Service`, `Payment Service`).
4. **Microservice Execution**
   Microservice processes requests, interacts with DB/event bus, and returns responses.
5. **Response Returned**
   API Gateway aggregates responses if required, applies response transformations, and returns to clients.

Security Considerations

- **Transport Encryption:** All external and internal communication is TLS 1.2+/mTLS enabled.
- **OAuth2/OpenID Connect:** Access tokens validated at the gateway.
- **WAF Protection:** OWASP Top-10 covered at the edge.
- **Rate Limiting:** Prevent brute-force and abusive traffic.
- **API Keys & Contracts:** Third-party partners (theatre systems, payment gateways) integrate via signed API contracts.

Deployment Considerations

- **Multi-Geo API Gateways**: Each geo hosts its own API Gateway cluster for latency optimization.
- **Blue/Green & Canary Deployments**: API Gateway supports routing specific traffic percentages to new versions.
- **Autoscaling**: Horizontal scaling based on request throughput (RPS).
- **Resiliency**: Circuit breakers and fallback responses for degraded services.

Example API Gateway Policies

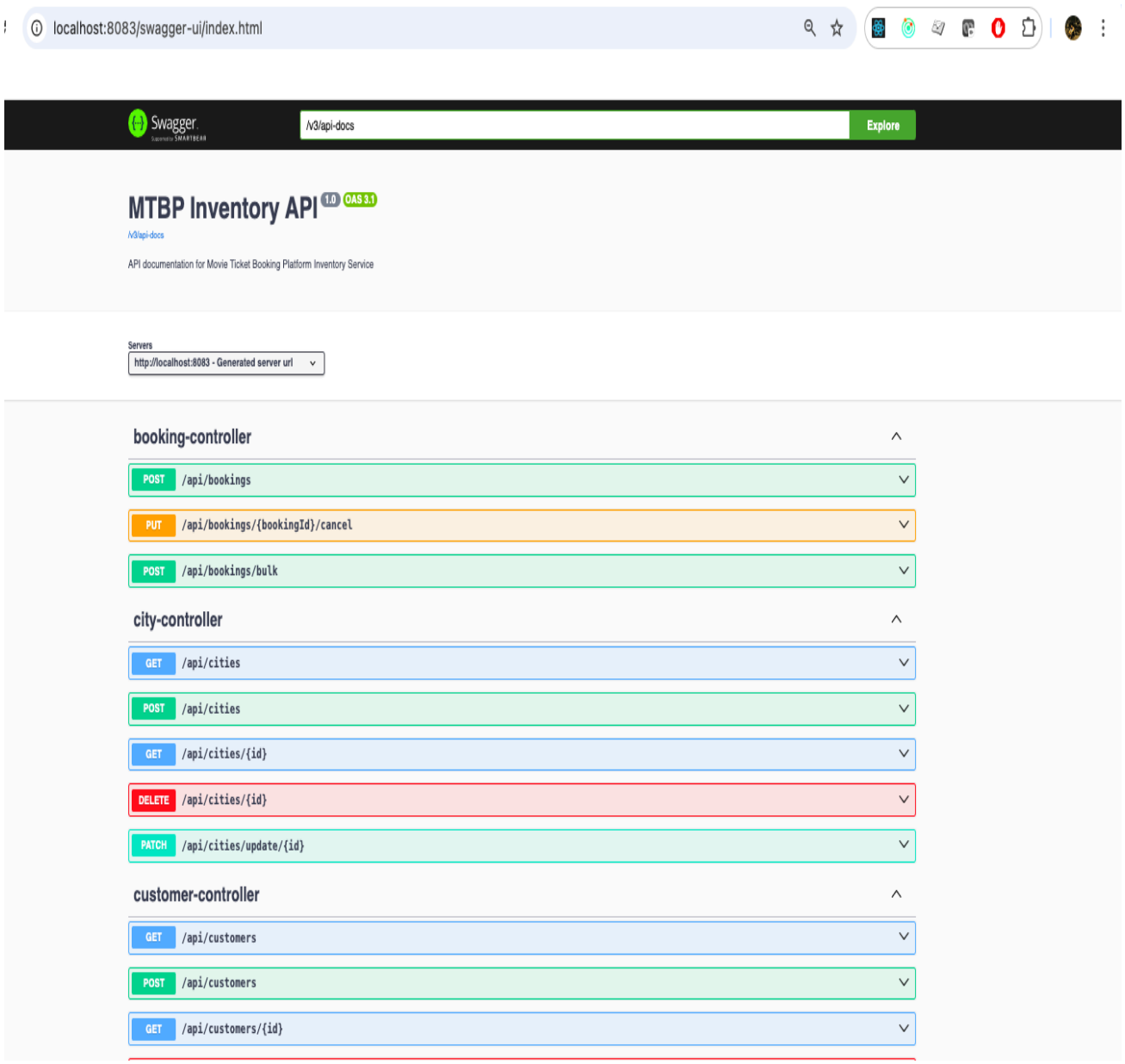
Policy	Description
Auth Policy	Enforce JWT validation via OKTA; reject expired/invalid tokens.
Quota Policy	Limit 100 requests/sec per user in peak hours.
Transformation Policy	Map external `/book` request into internal `BookingService.createBooking()` payload.
Observability Policy	Inject trace IDs into every request, push logs to Loki, metrics to Prometheus.

7.2 Microservices: Service-Level Responsibilities

Service	Responsibilities
Customer Service	Manage user profiles, authentication/authorization (via OKTA), loyalty points, preferences, registration/login, and profile updates.
Theatre Partner Service	Onboard theatre partners, manage theatre metadata, show schedules, seat layouts, and validate partner data.
Inventory Service	Maintain real-time seat availability, manage seat reservations, publish inventory events (Kafka), handle seat blocking and release.
Booking Service	Handle booking creation, updates, cancellations, seat allocation, event publishing (BookingCreated/Cancelled), and booking reconciliation.
Payment Service	Process payments and refunds, integrate with third-party payment gateways, maintain payment audit logs, handle transaction reconciliation.
API Gateway	Route requests to microservices, enforce authentication/authorization, throttling, and rate-limiting.
Load Balancer	Distribute incoming traffic, perform health checks, and ensure high availability.
Observability Services	Prometheus collects metrics, Loki collects logs, Grafana dashboards provide visualization.

7.2 API Contracts (High-Level Examples)

We can refer Swagger OpenAPI Documentation also for this, going forward:
https://mbtp.com/swagger-ui/index.html



7.2.1 Customer Service

- `POST /api/v1/customer` : Register new user.
- `GET /api/v1/customer/{id}`: Fetch user profile.
- `PATCH /api/v1/customer/{id}`: Update user profile.
- `GET /api/v1/customer`: Fetch list of users profile.
- `DELETE /api/v1/customer/{id}`: Delete user profile.

7.2.2 Booking Service

- `POST /api/v1/booking/customer/{id}/show/{id}`: Create new booking (seat selection + payment reference).
- `PUT /api/v1/booking/{id}/cancel`: Cancel booking.
- `POST /api/v1/bookings/bulk/numberOfTicketsReq/#/customer/{id}/show/{id}`: Bulk Booking.

7.2.3 Show Service

- `POST /api/v1/show`: Create a show.
- `PATCH /api/v1/shows/update/{showId}`: Update show details
- `DELETE /api/v1/show/{showId}`: Delete show by Id.

7.2.4 Payment Service

- `POST /api/v1/payment/charge`: Initiate payment.
- `POST /api/v1/payment/refund`: Refund a transaction.
- `GET /api/v1/payment/{id}`: Retrieve payment status.

> **Note:** Event-driven interactions are handled via Kafka topics, e.g., `BookingCreated`, `InventoryUpdated`, `PaymentSuccess`, `PaymentFailure`.

7.3 Data Modeling (High-Level)

Entity	Key Attributes	Related Service
Customer	`id`, `name`, `email`, `mobile`, `loyaltyPoints`	Customer Service
Theatre	`id`, `name`, `location`, `shows`	Theatre Partner Service
Show	`id`, `movieId`, `theatreId`, `startTime`, `endTime`, `seatMap`	Theatre/Inventory Service
Booking	`id`, `customerId`, `showId`, `seats`, `status`, `paymentId`	Booking Service
Payment	`id`, `bookingId`, `amount`, `status`, `gatewayResponse`	Payment Service

Database Considerations:

- **PostgreSQL** for relational consistency.
- **Caffeine** for In-Memory & **Redis** for distributed caching.
- **Partitioning / Sharding** by theatre or city for scalability.
- **Read replicas** for read-heavy operations like search.

7.4 Messaging & Eventing (Kafka)

- **Kafka Topics & Event Flows**
  - `InventoryUpdated`: Inventory Service → Booking Service (seat changes).
  - `BookingCreated`: Booking Service → Inventory Service (reserve seats).
  - `PaymentSuccess` / `PaymentFailure`: Payment Service → Booking Service (update booking status).
- **Rationale:** Decouples services, supports eventual consistency, and allows horizontal scaling.

7.5 Security Considerations

- **Authentication / Authorization**
  - OAuth2 / OpenID Connect via OKTA for all users (customers, admins, partners).
  - JWT-based access tokens for inter-service calls.
- **Data Protection**
  - Sensitive data encrypted at rest (AES-256).
  - TLS 1.2+ for in-transit data.
- **Compliance**
  - PCI-DSS for card payments.
  - Audit logs for all financial and critical operations.

7.6 Observability & Monitoring

- **Metrics**: Number of bookings, payment transactions, seat reservations, API latency per endpoint.
- **Logging**: Centralized structured logs via Loki; correlation IDs for tracing booking/payment workflows.
- **Tracing**: Distributed tracing using OpenTelemetry across microservices.
- **Alerts**: SLA breaches, error rates, failed payments, and high-latency events.
- **Runbooks**: Documented operational steps for incidents like DB failure, payment gateway outage, Kafka broker down.
- **Monitoring**: Grafana Dashboards..

7.7 Resilience & Fault Tolerance

- **Circuit breakers** and fallback strategies at service level.
- **Retries with exponential backoff** for transient errors.
- **Multi-region active-active deployments** to handle zone/region failures.
- **Compensating transactions** for eventual consistency in booking/payment flows.

7.8 CI/CD & Deployment

- **Pipeline:** GitHub Actions / Jenkins → Build → Test → Containerize → Deploy to Kubernetes (Blue/Green or Canary).
- **Observability Integration:** Prometheus, Loki, and Grafana monitoring included in deployment pipelines.
- **Rollback Strategy:** Versioned Helm charts with automated rollback on health check failures.

8. Platform Provisioning & Hosting

This section addresses **how MTBP is hosted, scaled, and provisioned**, ensuring high availability, performance, and maintainability.

8.1 Cloud / Hybrid Strategy

Options Considered:

- **Public Cloud (AWS / GCP / Azure):** Fully managed services for databases, Kubernetes, messaging, storage, and observability.
- **Hybrid Cloud:** Core sensitive services (e.g., payment reconciliation, financial audit data) can be hosted on private cloud/data center for compliance, while customer-facing services are on public cloud for scalability.

Rationale:

- Cloud-native architecture allows **elastic scaling** based on peak load (e.g., blockbuster movie releases).
- Managed services reduce operational overhead for database, Kafka, and monitoring.
- Hybrid strategy allows compliance with **regional data sovereignty requirements**.

8.2 Region & Multi-Geo Considerations

- **Multi-region deployment:** Active-active setup across regions to support availability SLA of 99.99%.
- **Data locality:** Store user and booking data regionally to reduce latency and comply with local regulations.
- **Failover & Disaster Recovery:**
    - Cross-region replication for critical databases (PostgreSQL replicas).
    - Multi-region Kafka cluster for event replication.
    - Load balancer with geo-routing to direct users to the nearest healthy region.

Rationale: Ensures **low-latency access**, compliance with **data localization laws**, and **resilient disaster recovery**.

8.3 Sizing & Capacity Planning

- **Microservices:**
  - Autoscaling based on **CPU/memory usage** and **RPS per service**.
  - Typical baseline: 3–5 replicas per service per region; scale to 50+ replicas for peak events.
- **Databases:**
  - PostgreSQL instances with **read replicas** and **partitioned tables** (by theatre or city) to handle high transaction throughput.
  - Estimate: 100K RPS peak globally → scale writes with sharding, reads via replicas.
- **Messaging (Kafka):**
  - 3–5 broker clusters per region, with replication factor ≥ 3 for durability.
  - Topics partitioned based on theatre or city for parallel consumption.
- **Storage & Cache:**
  - Object storage (S3/GCS/Azure Blob) for media content (posters, trailers).
  - Redis or Memcached for caching frequently accessed data (movies, showtimes, seat maps).

Rationale: Ensures platform can handle **peak loads**, maintain SLA, and prevent bottlenecks during high-demand periods.

8.4 Infrastructure as Code (IaC)

- **Tools:** Terraform / CloudFormation / Pulumi for provisioning cloud resources.
- **Benefits:**
  - Version-controlled infrastructure for repeatable deployments.
  - Automated setup of clusters, networking, databases, messaging, observability.
  - Supports multi-region and multi-environment (dev, staging, prod) setups.
- **Implementation Example:**
  - Define Kubernetes clusters, node pools, auto-scaling policies, VPC/subnets, managed DB instances, and Kafka clusters in IaC scripts.
  - CI/CD pipelines trigger infrastructure provisioning before deploying services.

Rationale: Enables **consistent, repeatable, and auditable deployments**, reducing human error and speeding up multi-region provisioning.

9. Data Management & Transactions

This section explains how **data is structured, managed, and protected** across MTBP microservices, ensuring **consistency, availability, and resilience**.

9.1 Data modelling (ER / high-level schema)

High-Level Data Entities:

Entity	Key Attributes	Description
Customer	`id`, `name`, `email`, `phone`, `loyaltyPoints`	End customer information and preferences
Theatre	`id`, `name`, `location`, `partnerId`, `screens`	Theatre metadata and partner association
Screen / Show	`id`, `theatreId`, `movieId`, `startTime`, `endTime`, `seatMap`	Represents show schedules and seating layouts
Booking	`id`, `customerId`, `showId`, `seatNumbers`, `status`, `paymentId`	Tracks bookings, status (confirmed, cancelled, pending)
Payment	`id`, `bookingId`, `amount`, `currency`, `status`, `gatewayResponse`	Payment transactions and audit trail
Inventory	`showId`, `seatNumber`, `status`	Real-time seat availability and reservation tracking

Notes:

- **PostgreSQL** is used for relational consistency.
- **Partitioning / Sharding:** by theatre, city, or show to handle high volume and reduce contention.
- **Indexes:** on frequently queried attributes like `showId`, `customerId`, and `status` for performance.

9.2 Transactional Guarantees

- **Booking & Inventory Updates:**
  - Use **ACID transactions** within a single service (e.g., Booking Service writing to Booking DB and Inventory Service writing to Inventory DB).
  - **Two-phase commit avoided** across microservices to prevent blocking; rely on event-driven compensation.
- **Payment Processing:**
  - Payment service guarantees **idempotent operations** to prevent duplicate charges.
  - Transaction records are persisted before invoking external payment gateway APIs.

Rationale: Ensures **data correctness** while maintaining **high throughput** and scalability.

9.3 Eventual Consistency

- **Asynchronous Messaging (Kafka):**
  - Booking Service publishes `BookingCreated` → Inventory Service consumes → updates seat reservations.
  - Payment Service publishes `PaymentSuccess/Failure` → Booking Service updates booking status.
- **Compensating Transactions:**
  - If payment fails after booking, booking is cancelled and inventory is released.
  - Ensures **system state converges to a consistent outcome** without blocking all services.

Rationale: Supports **high concurrency**, **scalable microservices**, and **resilient operations** during peak loads.

9.4 Backups & Archival

- **Backups:**
  - Daily full backups of all PostgreSQL databases.
  - Incremental backups every 15 minutes for high-transaction tables (Booking, Payment, Inventory).
  - Cross-region backup storage for disaster recovery.
- **Archival:**
  - Older transactional data (>1 year) moved to cost-effective object storage (S3/GCS/Azure Blob) for audit and analytics.
  - Archived data is still queryable via analytics pipelines.
- **Retention Policies:**
  - Payment & booking history: 7 years (for compliance and audits).
  - Logs and observability data: 90 days in hot storage, >1 year in cold storage.

10. COTS & Enterprise Systems

MTBP relies on **trusted third-party systems and tools** for identity management, payment processing, observability, secrets, and CI/CD. This ensures **compliance, reliability, and maintainability**.

10.1 Identity & Access Management (IAM)

- **System:** OKTA (or equivalent enterprise IAM)
- **Purpose:** Authentication, authorization, and user lifecycle management.
- **Features:**
    - OAuth2 / OpenID Connect support for web, mobile, and API access.
    - Role-Based Access Control (RBAC) for admins, partners, and internal services.
    - Single Sign-On (SSO) for employees and theatre partners.
    - Audit logs for all identity-related operations.
- **Rationale:** Centralized IAM ensures **security, regulatory compliance, and simplified user management**.

10.2 Payment Gateways

- **Systems:** Stripe, Razorpay, PayU, or other PCI-DSS compliant providers.
- **Purpose:** Secure payment processing and reconciliation.
- **Features:**
    - Multi-currency and international payment support.
    - Refunds, chargebacks, and recurring payments.
    - Integration via REST APIs and webhooks for real-time updates.
    - Payment audit logs for compliance.
- **Rationale:** Outsourcing payments reduces PCI scope and leverages enterprise-grade reliability.

10.3 Observability & Monitoring

- **Systems:** Prometheus (metrics), Loki (logs), Grafana (dashboards)
- **Purpose:** End-to-end observability of all services, events, and infrastructure.
- **Features:**
    - Service-level and business-level metrics collection.
    - Centralized logging with correlation IDs.
    - Distributed tracing for booking/payment flows.
    - Alerts and automated runbooks for incident response.
- **Rationale:** Enables **proactive monitoring, SLA enforcement, and rapid troubleshooting**.

10.4 Secret & Configuration Management

- **Systems:** HashiCorp Vault, AWS Secrets Manager, or equivalent.
- **Purpose:** Secure storage and retrieval of credentials, API keys, and certificates.
- **Features:**
    - Dynamic secrets for databases and APIs.
    - Role-based access to secrets.
    - Automatic secret rotation and auditing.
- **Rationale:** Prevents accidental exposure of sensitive information and ensures compliance.

10.5 CI/CD & DevOps

- **Systems:** GitHub Actions, Jenkins, GitLab CI, ArgoCD, or equivalent.
- **Purpose:** Automate build, test, containerization, and deployment pipelines.
- **Features:**
    - Multi-environment (dev, staging, prod) pipelines.
    - Automated unit, integration, and load testing.
    - Blue/Green or Canary deployments to minimize downtime.
    - Integration with IaC (Terraform/CloudFormation) for repeatable provisioning.
- **Rationale:** Ensures **rapid, reliable, and auditable deployments**, supporting multi-region operations.

10.6 Enterprise Resource Planning (ERP)

- **Systems:** SAP / Oracle ERP / NetSuite (optional depending on scale)
- **Purpose:** Manage financials, accounting, and partner/vendor operations.
- **Integration:**
    - Payment reconciliation reports exported from Payment Service.
    - Theatre partner onboarding and commission management.
    - Automated journal entries and compliance reporting.
- **Rationale:** Provides **enterprise-grade financial management and auditability** without reinventing the wheel.

11. CI/CD, Release Management & Rollout Strategy

This section outlines the **end-to-end DevOps and release process** for MTBP, ensuring **high-quality deployments, minimal downtime, and controlled feature releases**.

11.1 Build Pipeline with Quality Gates

- **CI/CD Pipeline Tools:** GitHub Actions, Jenkins, GitLab CI, or equivalent.
- **Pipeline Steps:**
    1. **Code Checkout & Compilation:** Build microservices (Java Spring Boot) and front-end artifacts (React/Angular).
    2. **Unit & Integration Testing:** Run automated JUnit / Selenium / Cypress tests.
    3. **Static Code Analysis:**
        - Code quality: SonarQube
        - Security: OWASP Dependency Checks, CodeQL SAST scans
    4. **Containerization:** Build Docker images for all microservices.
    5. **Artifact Storage:** Push images to private container registry.
    6. **Deployment to Test/Staging:** Automated deployment to Kubernetes test clusters.
    7. **Acceptance Testing & Performance Validation:** Load testing to validate SLA targets.
    8. **Quality Gates:** Only pass to production if:
        - All tests pass
        - Code quality meets thresholds
        - Security checks are cleared

Rationale: Ensures only **high-quality, compliant code** reaches production.

11.2 Deployment Strategy

- **Blue-Green Deployment:**
    - Maintain two identical production environments (Blue & Green).
    - Deploy new version to inactive environment.
    - Switch traffic via Load Balancer once validation passes.
    - Rollback possible by reverting traffic to the previous environment.

- **Canary Deployment:**
    - Deploy new version to a small subset of users/services.
    - Monitor performance, logs, and errors.
    - Gradually increase traffic if metrics are within SLA.
- **Feature Flags:**
    - Enable/disable new features dynamically without redeploying.
    - Useful for A/B testing or gradual rollout to select geographies or customer segments.

Rationale: Minimizes user impact, allows safe testing, and provides fast rollback if issues occur.

11.3 Release Across Geos

- **Multi-Region Deployment:**
    - Staged rollout by region to monitor local SLAs and performance.
    - Regional traffic routing via Geo-DNS or Load Balancer configuration.
- **Data Migration & Synchronization:**
    - Use database replication and Kafka for cross-region data consistency.
- **Rollback & Observability:**
    - Rollback plans per region.
    - Metrics, tracing, and logging monitored for anomalies before expanding rollout.

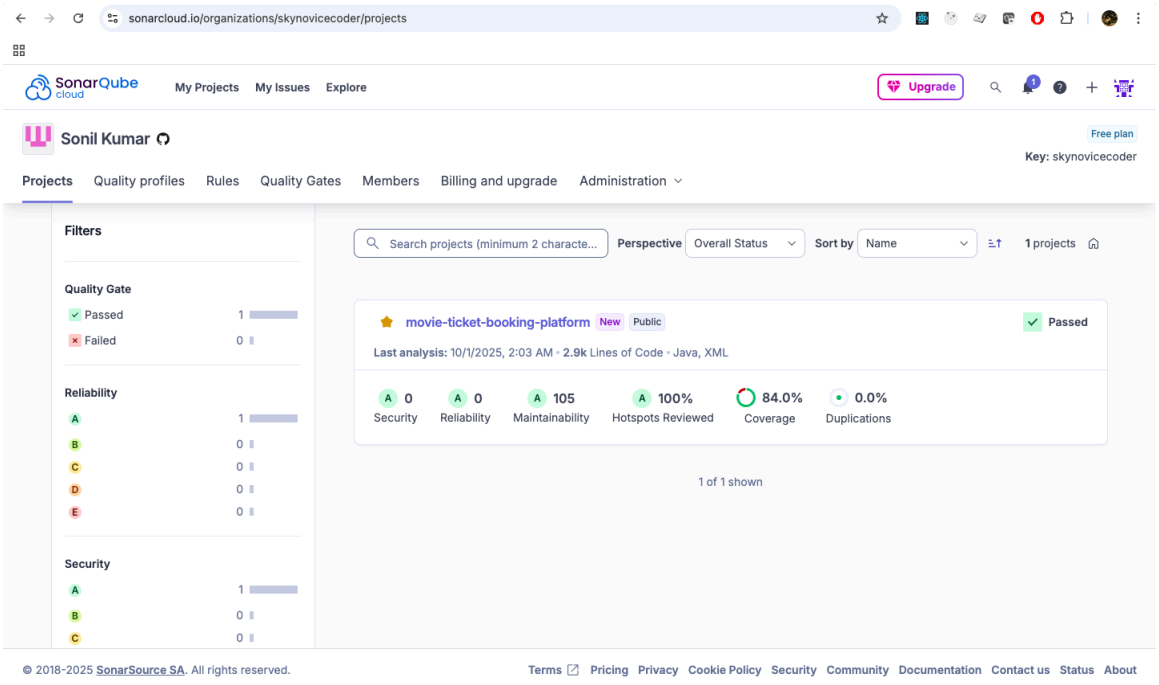
Rationale: Ensures controlled release, compliance with local regulations, and reliable cross-region operations.

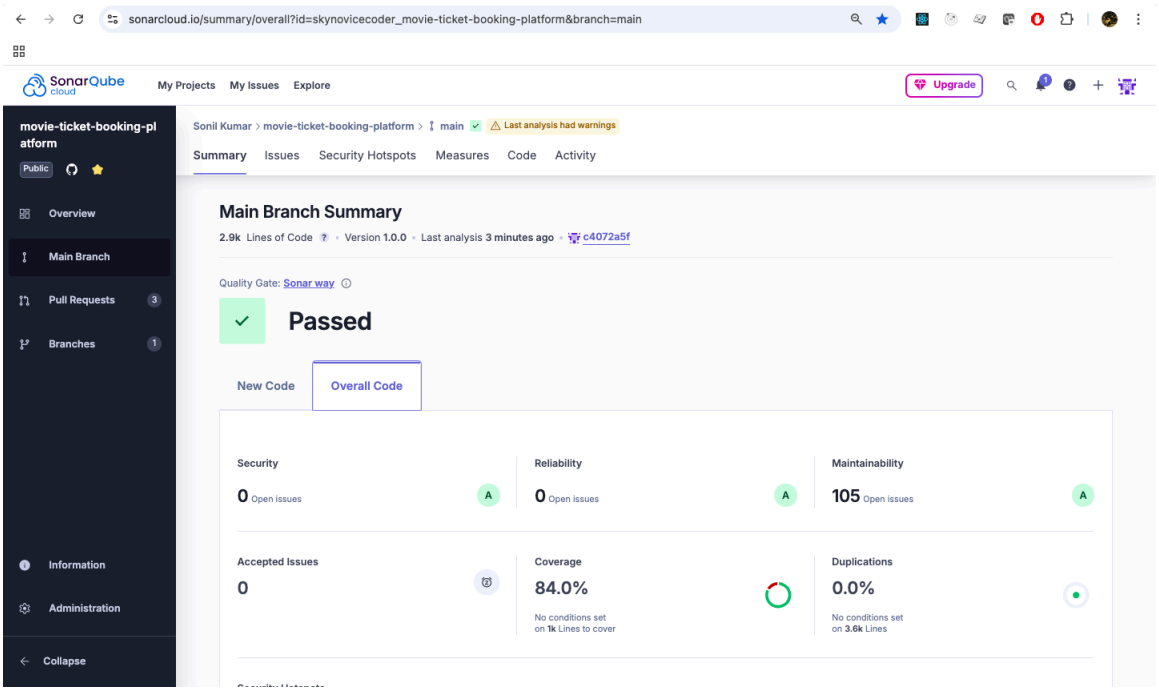
11.4 CI/CD & Release Observability

- **Automated Alerts:** For failed builds, failing tests, or SLA breaches in canary deployments.
- **Monitoring Integration:** Prometheus & Grafana dashboards track:
    - Deployment health
    - Service response times
    - Error rates
    - Traffic patterns during rollout
- **Audit & Reporting:** Every deployment and feature flag change is logged for compliance.

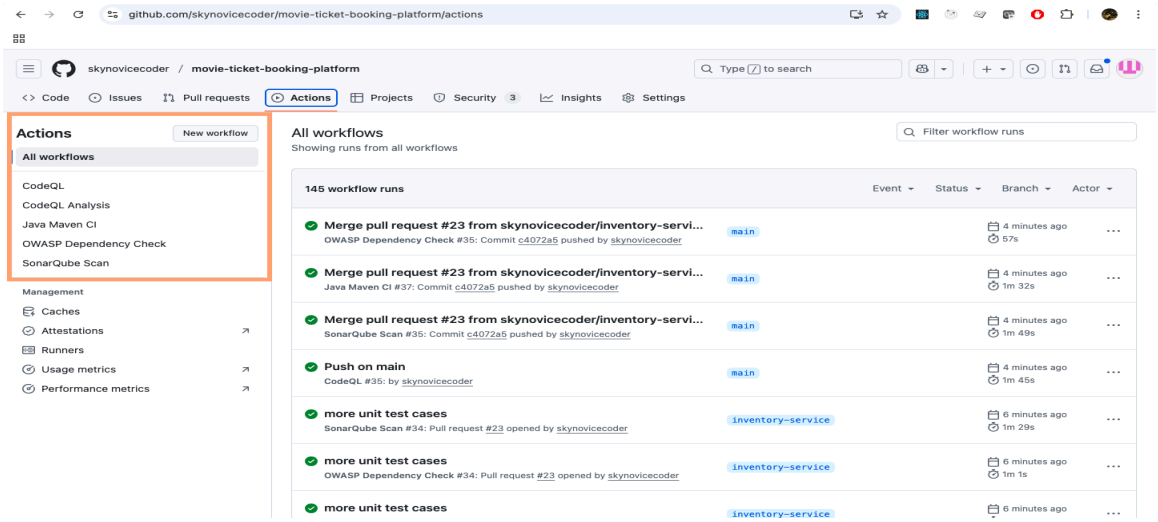
11.5 Sonarqube Cloud Link

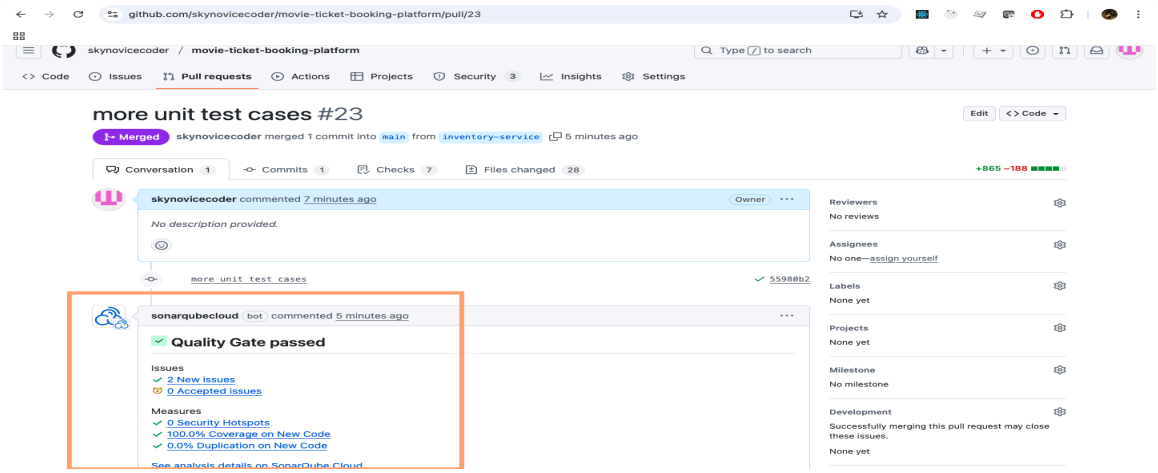
https://sonarcloud.io/project/overview?id=skynovicecoder_movie-ticket-booking-platform

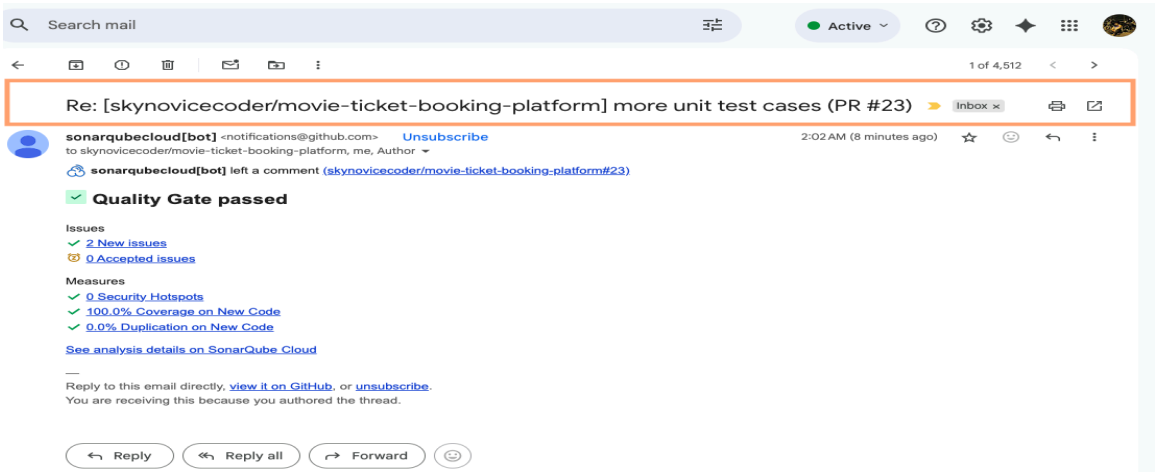




11.6 GitHub Actions : WorkFlows







12. Scalability, Availability & Resilience

This section describes the **strategies and architectural patterns** implemented to ensure MTBP **can handle high load, recover from failures, and maintain SLA compliance (99.99% uptime)**.

12.1 Horizontal Scaling

- **Microservices:**
  - Deployed as **stateless containers** in Kubernetes, allowing autoscaling based on CPU, memory, or request metrics.
  - Example: Booking Service scaled up during blockbuster releases to handle up to **100,000 RPS globally**.
- **Databases:**
  - PostgreSQL reads replicas for read-heavy operations (search, seat availability).
  - Partitioned or sharded tables for write-heavy services (Booking, Payment, Inventory).
- **Caching:**
  - Redis/Memcached for frequently accessed data (showtimes, seat maps) to reduce DB load.

Rationale: Horizontal scaling ensures **elastic capacity** to meet peak loads without service degradation.

12.2 Partitioning / Sharding

- **Booking & Inventory DBs:**
  - Sharded by **theatreId or city** to distribute load and prevent hotspots.
- **Kafka Topics:**
  - Partitioned by theatre or showId for parallel consumption and high throughput.
- **Session / Cache Data:**
  - Distributed caches keyed by region or user segment to avoid bottlenecks.

Rationale: Data partitioning reduces contention, supports concurrency, and improves system throughput.

12.3 Circuit Breakers, Retries & Resilience Patterns

- **Circuit Breakers:**
  - Protect downstream services from cascading failures.
  - Example: Payment Service failures temporarily open the circuit to prevent Booking Service overload.
- **Retries with Exponential Backoff:**
  - Handle transient errors in inter-service calls or external APIs (payment gateway, partner APIs).
- **Fallback Mechanisms:**
  - Graceful degradation if services are unavailable (e.g., show alternate theatre options).
- **Bulkhead Isolation:**
  - Service instance isolation to prevent failures from impacting unrelated workflows.

Rationale: Ensures **high reliability and graceful degradation** under partial failure conditions.

12.4 Disaster Recovery Strategy

- **Multi-Region Deployment:**
  - Active-active regions for high availability.
  - Automatic failover using Load Balancer / Geo-DNS in case of regional outages.
- **Data Replication:**
  - PostgreSQL cross-region replication for critical tables.
  - Kafka multi-broker replication across regions.
- **Backup & Restore:**
  - Automated daily full backups + incremental backups every 15 minutes.
  - Retention and archival policies to comply with audit requirements.
- **Runbooks & Automated Recovery:**
  - Predefined runbooks for service failures, DB failover, or Kafka broker outage.
  - Automation scripts for restoring service with minimal downtime.

Rationale: Guarantees **data durability, SLA compliance, and business continuity** during disasters.

13. Performance & Capacity Planning

This section outlines the **strategy for testing, measuring, and planning capacity** to ensure MTBP handles both normal traffic and peak demand with high availability and performance.

13.1 Performance Tests

- **Types of Tests:**
    - **Load Testing:** Simulate normal and peak user traffic to validate service response times.
        - Example: Search API 95th percentile < 300ms.
        - Booking flow end-to-end < 1.5s under normal load.
    - **Stress Testing:** Push services beyond expected peak to identify breaking points and bottlenecks.
    - **End-to-End Flow Testing:** Simulate real user journeys (search → seat selection → booking → payment).
    - **Integration Testing:** Ensure payment gateways, Kafka messaging, and partner APIs perform under load.
- **Tools:** JMeter, Gatling, Locust, or k6 for HTTP and event-driven testing.

Rationale: Ensures **platform performance meets SLAs** and identifies potential bottlenecks before production release.

13.2 Capacity Plans for Big Premieres

- **Peak Load Estimation:**
    - Historical data + marketing projections used to forecast peak RPS.
    - Example: Blockbuster releases may generate **~100,000 RPS globally**, concentrated in top cities.
- **Service Scaling:**
    - Horizontal autoscaling for microservices based on CPU, memory, or request metrics.
    - Pre-warm additional Kubernetes pods for anticipated peak.
- **Database Scaling:**
    - Sharded and partitioned DBs to distribute write-heavy traffic.
    - Read replicas for read-intensive operations like movie searches and showtimes.

- **Caching Strategies:**
    - Redis/Memcached to serve frequently requested data (movie listings, seat maps).
    - Preload caches before anticipated peak.
- **Messaging & Event Handling:**
    - Kafka partitions tuned for parallel consumption.
    - Producer/consumer throughput monitored and scaled to avoid bottlenecks.
- **Observability & Alerts:**
    - Dashboards to monitor latency, error rates, and queue depth.
    - Auto-trigger alerts for thresholds exceeding SLA during premieres.

Rationale: Ensures **predictable, reliable performance** even during extreme load conditions without impacting user experience.

14. Security, Compliance & Privacy

This section describes **security controls, compliance measures, and privacy practices** to safeguard customer data, payment information, and platform operations.

14.1 Regulatory Compliance

- **PCI-DSS (Payment Card Industry Data Security Standard):**
    - All cardholder data handled by Payment Service and third-party gateways.
    - No sensitive card data stored on MTBP databases; tokenization used.
    - Regular audits and logging of all payment operations.
- **GDPR / Data Privacy Compliance:**
    - Customer data processed in accordance with regional data protection laws.
    - Data subject rights enforced: access, deletion, and portability.
    - Region-specific data storage for EU and other regulated geographies.
- **Audit & Reporting:**
    - Centralized logging of security-sensitive operations (login, payment, booking).
    - Reports generated for compliance audits, data breaches, or internal reviews.

14.2 Transport & Data Encryption

- **In-Transit:**
  - TLS 1.2+ enforced for all client-service and inter-service communication.
  - Mutual TLS (mTLS) for sensitive microservice interactions (e.g., Booking ↔ Payment).
- **At-Rest:**
  - AES-256 encryption for databases, caches, and backups.
  - Encrypted storage for logs containing sensitive identifiers.
- **Key Management:**
  - Keys managed via **HashiCorp Vault / AWS KMS / Azure Key Vault**.
  - Automated key rotation and auditing enabled.

14.3 Identity & Access Management (IAM) Best Practices

- **Authentication:**
  - OAuth2/OpenID Connect via OKTA for all users (customers, admins, partners).
  - Multi-factor authentication (MFA) for admin and partner portal access.
- **Authorization:**
  - Role-Based Access Control (RBAC) at API and service layer.
  - Principle of least privilege enforced for service accounts and users.
- **Secrets Management:**
  - No credentials hard-coded in services.
  - Secrets fetched dynamically from a secure vault at runtime.

14.4 Penetration Testing & Security Reviews

- **Internal Security Testing:**
  - Automated SAST/DAST scans integrated into CI/CD pipelines.
  - Dependency vulnerability scanning (OWASP, CodeQL).
- **External Penetration Testing:**
  - Periodic third-party pen-tests on platform endpoints, APIs, and admin portals.
  - Remediation plan documented and executed before production releases.
- **Security Monitoring:**
  - Anomalous access patterns, failed login attempts, and suspicious activity logged and alerted.

15. Operational Readiness & Runbooks

This section defines **how MTBP is monitored, supported, and recovered** in case of failures or incidents, including **on-call responsibilities, playbooks, and communication procedures**.

15.1 On-Call & Incident Playbooks

- **On-Call Structure:**
  - 24/7 coverage split across SRE and Engineering teams.
  - Rotation schedule for **primary**, **secondary**, and **escalation** contacts.
  - Responsibilities defined per service (Booking, Payment, Inventory, Customer, Theatre Partner).
- **Incident Playbooks:**
  - Predefined step-by-step procedures for common incidents:
    - Payment Gateway failures
    - Kafka broker outages
    - Database failover or replica lag
    - Booking Service high-latency issues
  - Include **detection triggers, mitigation steps, and rollback procedures**.
  - Integrated with alerting systems (Prometheus Alerts, Grafana dashboards).

15.2 Root Cause Analysis (RCA) Templates

- **Purpose:** Standardized RCA for post-incident review and continuous improvement.
- **Template Sections:**
  - Incident Summary
  - Timeline of Events
  - Impact Analysis (users, revenue, systems)
  - Root Cause
  - Corrective & Preventive Actions
  - Lessons Learned
  - Approval & Distribution
- **Process:**
  - RCAs completed within 48 hours of incident resolution.
  - Shared with stakeholders and retained for compliance/audit purposes.

15.3 Status Page & Communication

- **Public/Private Status Page:**
  - Displays **real-time platform health**: booking service, payment processing, inventory, and partner integrations.
  - Provides **incident notifications** and historical uptime SLA metrics.
- **Communication Channels:**
  - Slack/MS Teams channels for internal alerts and escalations.
  - Email/SMS notifications for critical incidents.
  - Status updates posted to external stakeholders and partner theatres if affected.

15.4 Observability Integration for Operations

- **Metrics & Dashboards:**
  - Microservice-level metrics (request rates, error rates, latency).
  - Infrastructure metrics (CPU, memory, disk usage).
  - Business KPIs (booking count, payment success/failure).
- **Logging & Tracing:**
  - Centralized logging via Loki.
  - Distributed tracing for end-to-end booking/payment flows.
  - Correlation IDs to link user requests across services.
- **Alerting & Escalation:**
  - Threshold-based and anomaly detection alerts.
  - Automated escalation to on-call engineers with clear runbooks.

16. Cost Estimates & Licensing

This section provides a **high-level estimate of the costs** associated with running MTBP and identifies **opportunities for cost efficiency**.

16.1 Cloud Infrastructure Costs

- **Compute:**
  - Kubernetes clusters running microservices (Booking, Payment, Inventory, Customer, Theatre Partner).
  - Autoscaling ensures cost efficiency by scaling down during off-peak periods.
- **Databases:**
  - Managed PostgreSQL instances with read replicas.
  - Partitioning and sharding for optimized usage.

- **Messaging & Event Streaming:**
  - Managed Kafka clusters per region.
  - Cost depends on number of brokers, partitions, and replication factor.
- **Storage & Caching:**
  - S3/GCS/Azure Blob for media content and backups.
  - Redis/Memcached clusters for caching.

Estimation Approach:

- Based on **RPS, peak concurrency, and storage requirements**.
- Includes **multi-region deployment** for high availability.

16.2 COTS & Licensing Costs

System	Licensing Model	Estimated Cost Considerations
Identity (OKTA)	SaaS subscription per user / per app	Cost scales with number of admin/partner users and integrations
Payment Gateway	Transaction fees + gateway subscription	Depends on RPS, payment volume, and international transactions
Observability (Grafana, Loki, Prometheus)	Open source / Enterprise subscription	Enterprise edition recommended for SLA support & alerting
Secrets Management (Vault / KMS)	Subscription per instance or cloud-managed	Key management and rotation included
ERP / Financial Systems	Annual license + implementation	Optional depending on scale; used for partner reconciliation & accounting

16.3 Cost Optimization Strategies

1. **Autoscaling & Spot Instances:**
    - Use cloud autoscaling for microservices and worker nodes.
    - Use spot/preemptible instances for non-critical workloads (analytics, batch jobs).
2. **Right-Sizing Resources:**
    - Monitor actual CPU/memory usage and adjust instance types accordingly.
    - Avoid over-provisioning for low-traffic regions.
3. **Data Lifecycle Management:**
    - Archive cold data to cost-effective storage (S3 Glacier / Azure Cool Blob).
    - Retention policies aligned with compliance requirements.
4. **Consolidated Licensing:**
    - Enterprise-wide licenses for observability, CI/CD, and IAM to reduce per-instance costs.
5. **Cloud Savings Plans / Reserved Instances:**
    - Prepay or commit to long-term instances for predictable workloads to reduce costs.

17. Risk Assessment & Mitigation

This section provides a structured view of **potential risks** to MTBP and outlines **proactive measures** to minimize impact on business and technical operations.

17.1 Top Risks

Risk Category	Description	Potential Impact
Traffic Spikes / Peak Loads	Sudden surge in bookings during blockbuster releases or promotional campaigns.	Service degradation, high latency, failed bookings, SLA breaches.
Payment Gateway Failures	Outages or delays in third-party payment providers.	Payment failures, lost revenue, customer dissatisfaction.
Partner Integration Issues	Theatre API downtime or incorrect inventory data.	Booking errors, seat overbooking, poor user experience.
Data Loss / Database Outage	DB corruption, misconfiguration, or region failure.	Loss of bookings, payment records, and inventory data.
Security Breaches / Data Leakage	Unauthorized access, compromised credentials, or malware.	Customer data exposure, PCI/GDPR compliance violations, reputational damage.
Infrastructure Failures	Network outages, cloud provider disruptions, or Kubernetes cluster failures.	Partial or full service unavailability, SLA breaches.
Operational Errors	Misconfigurations, deployment issues, or human error.	Service downtime, failed releases, degraded performance.

17.2 Mitigation Plans

Risk	Mitigation Strategy
Traffic Spikes / Peak Loads	Horizontal scaling, partitioned DBs, caching, pre-warmed pods, load testing, autoscaling policies.
Payment Gateway Failures	Implement retries with exponential backoff, fallback to alternate gateways, asynchronous reconciliation, alerting & runbooks.
Partner Integration Issues	Circuit breakers, retries, monitoring of partner APIs, validation of inventory updates, compensating transactions.
Data Loss / Database Outage	Multi-region replication, automated backups, incremental backups, disaster recovery plans, failover procedures.
Security Breaches / Data Leakage	IAM best practices, encryption in transit & at rest, regular penetration testing, secrets management, audit trails.
Infrastructure Failures	Multi-region deployment, load balancer failover, Kubernetes self-healing, automated monitoring & alerting.
Operational Errors	Standardized CI/CD pipelines, QA/staging validation, rollback strategies, runbooks, and SRE oversight.

18. Stakeholder Management & Decision Logs

This section outlines the **approach to engaging stakeholders**, documenting **architectural and product decisions**, and ensuring alignment between **business, product, and technical teams**.

18.1 Architectural Decision Records (ADRs) with Rationale

- **Purpose:** Document **key technical and architectural decisions**, along with alternatives considered and trade-offs.
- **Structure of an ADR:**
    - **Title:** Brief description of the decision.
    - **Context:** Why the decision is needed.
    - **Decision:** Chosen solution or approach.
    - **Alternatives Considered:** Other approaches with pros/cons.
    - **Consequences:** Impact on architecture, operations, costs, and users.
    - **Status:** Proposed, Accepted, Deprecated, or Superseded.

- **Example ADRs for MTBP:**
    - Choice of **PostgreSQL** over NoSQL for transactional consistency.
    - Decision to use **Kafka for event-driven communication**.
    - Adoption of **OKTA for centralized identity management**.
    - Implementation of **Blue/Green & Canary deployments** for release management.

Rationale: ADRs provide **traceable and auditable decision-making**, especially critical in multi-stakeholder enterprise environments.

18.2 Stakeholder Engagement Model

- **Key Stakeholders:**
    - Product Owner
    - Enterprise & Solution Architects
    - Engineering Leads (Frontend, Backend, Data, Platform)
    - QA & SRE teams
    - Security & Compliance
    - Theatre Partnership Managers
    - Payment Gateway / Vendor teams
    - Regional Operations Teams

- **Engagement Approach:**
    - **Regular Steering Meetings:** Align business objectives, prioritize features, and approve architectural changes.
    - **Design Reviews:** Solution Architects present high-level and detailed designs for review and feedback.
    - **Decision Logs & ADR Repository:** All technical decisions are documented, version-controlled, and accessible to stakeholders.
    - **Change Control Board (CCB):** Reviews major changes impacting SLA, compliance, or multi-region deployments.
    - **Communication Channels:**
        - Slack/Teams for real-time operational updates
        - Email for formal announcements
        - Shared dashboards for observability, performance, and release status
- **Escalation Model:**
    - Tiered escalation for critical issues impacting platform availability, payments, or data integrity.
    - Defined SLA for stakeholder notifications and decision closure.

19. Appendices

This section provides **supporting information, glossary and contact points** to aid understanding and implementation of MTBP.

19.1 Glossary

Term	Definition
MTBP	Movie Ticket Booking Platform
RPS	Requests Per Second, used to measure system load
C4	Context, Container, Component, and Code diagrams framework
ADR	Architectural Decision Record
SLA	Service Level Agreement
PCI-DSS	Payment Card Industry Data Security Standard
GDPR	General Data Protection Regulation
RBAC	Role-Based Access Control
CI/CD	Continuous Integration and Continuous Deployment

19.2 Key Contacts

Role	Contact	Responsibilities
Product Owner	[Name/Email]	Feature prioritization, business requirements
Enterprise Architect	[Name/Email]	High-level architecture guidance, ADR oversight
Solution Architect	[Name/Email]	Technical design, component-level review
Engineering Leads	[Name/Email]	Implementation guidance, code quality enforcement
SRE / QA	[Name/Email]	Production readiness, testing, incident response
Security & Compliance	[Name/Email]	Audits, security policies, pen-testing
Theatre Partnership Managers	[Name/Email]	Partner onboarding, API integration
Payment Gateway Vendors	[Name/Email]	Payment integration, SLA coordination

19.2 Important Links

Architecture & Design Artifacts

C4 Architecture Diagrams Repository
https://github.com/skynovicecoder/mtbp-c4-diagrams
Contains:

- C4 Context, Container, Component, and Code-level diagrams
- UML Diagrams (ER, Sequence, Class)
- Architecture Decision Records (ADRs)

Code Base

https://github.com/skynovicecoder/movie-ticket-booking-platform
Contains microservices source code for:

- Backend
- Frontend
- Infrastructure & DevOps

SonarQube Security Scan Links:
https://sonarcloud.io/project/overview?id=skynovicecoder_movie-ticket-booking-platform

Document Repository
https://github.com/skynovicecoder/mtbp-c4-diagrams/tree/main/docs

Design & Governance Documents
Contains:

- High-Level & Low-Level Design Documents
- CI/CD Flow Diagrams
- Release Notes & Runbooks
- Compliance Checklists (PCI-DSS, GDPR, ISO27001)

Operational Docs
Contains:

- Monitoring Dashboards & On-call Schedules
- RCA Templates
- Incident Response Playbooks