Functional Decomposition of an Event Ticket Booking System

Assignment 1

UX Design Studio – IV
Course Code: UXD 205

Student Name:
Pujan Aggarwal

Enrollment Number:
A021142924039

Submission Date:
15 December 2025

2. Introduction

The chosen system for this assignment is an **Event Ticket Booking Application**, similar in scope and complexity to platforms such as BookMyShow. The system enables users to discover events, select venues and showtimes, choose seats, complete payments, receive tickets, and manage bookings after purchase. It also supports backend processes such as inventory control, pricing management, payment verification, fraud detection, and automated notifications.

Functional decomposition is an important technique for understanding complex systems because it breaks a large system into smaller, logically related components. Instead of focusing on screens or interface elements, functional decomposition focuses on **what the system does** and **how responsibilities are distributed** across layers. This approach helps designers and engineers identify hidden logic, system dependencies, failure points, and automation requirements early in the design process.

The scope of this decomposition is limited to the **digital ticket booking platform**, including user-facing features and backend system logic required to support them. Physical venue operations, hardware scanners, and third-party infrastructure beyond system integration points are excluded. The analysis focuses on functional behavior rather than visual design or implementation details.

---

3. Methodology

The functional decomposition was approached by first identifying the primary goals a user attempts to accomplish within the system, such as discovering events, booking tickets, making payments, and managing bookings. These high-level goals were translated into major functional domains, which form the first level of the decomposition.

Each primary function was then broken down progressively into more detailed sub-functions. The decomposition was carried out across five distinct layers. **Level 1** represents primary functional buckets that define broad system responsibilities. **Level 2** captures user-visible sub-functions within each bucket. **Level 3** represents detailed functional capabilities that support those sub-functions. **Level 4** contains micro-components and operational steps required to execute features. **Level 5** consists of conditional logic and system-only operations that handle automation, validation, error recovery, and reliability.

Features were categorized based on responsibility rather than screen placement. User-facing capabilities were placed in higher layers, while automation, validation, and background processes were pushed to deeper layers. Shared logic such as inventory synchronization, fraud detection, and payment verification was centralized to avoid duplication and ensure consistency across the system.

Throughout the decomposition, special attention was given to concurrency, failure scenarios, edge cases, and backend automation. This ensured that the system model reflects real-world operating conditions such as high traffic, payment ambiguity, and partial user actions.

4. ACTION → FEATURE MAPPING TABLE

#	USER ACTION	TRIGGER	DESTINATION	SYSTEM REACTION
1	Clicks Sign Up	Button click	Registration	Display form
2	Enters Email/Submit	Form submit	Email Verify	Generate token
3	Clicks Verify Link	Link click	Activation	Mark verified
4	Enters Login Creds	Form submit	Authentication	Create session
5	Selects Remember Device	Checkbox	Device Manager	Store fingerprint
6	Searches Event Name	Search input	Search Index	Return results
7	Filters by City	Filter selection	Venue Manager	Apply filter
8	Selects Event Poster	Card click	Event Details	Load details
9	Clicks Show Time	Time selection	Seat Layout	Load grid
10	Clicks Seat	Seat click	Seat Lock	Lock, add cart
11	Clicks Add to Cart	Button click	Shopping Cart	Store selection
12	Clicks Checkout	Button click	Payment Pipeline	Calculate total
13	Enters Coupon Code	Apply click	Coupon Validator	Calculate discount
14	Selects Credit Card	Method selection	Payment Router	Display form
15	Enters Card Details	Form submit	Card Validator	Validate, encrypt
16	Payment Processing	API call	Payment Processor	Process charge
17	Payment Successful	Success event	Booking Confirm	Create booking
18	Payment Fails	Failure event	Retry Logic	Display error
19	Generates Ticket	Booking confirm	Ticket Generator	Create PDF
20	Sends Confirmation	Booking create	Email Service	Send ticket
21	Receives SMS	Booking event	SMS Gateway	Send link
22	Clicks Change Seats	Button click	Modification	Release locks
23	Initiates Cancel	Cancel click	Policy Checker	Calculate refund
24	Confirms Cancel	Confirmation	Refund Processor	Process refund
25	Requests Reminder	Remind click	Reminder Scheduler	Schedule reminder

#	USER ACTION	TRIGGER	DESTINATION	SYSTEM REACTION
26	Views My Bookings	Menu click	Booking Query	Display bookings
27	Downloads Ticket	Download click	Ticket Storage	Serve PDF
28	Transfers Ticket	Transfer click	Ticket Transfer	Send link
29	Publishes Event	Submit event	Event Manager	Store data
30	Uploads Seats	CSV upload	Venue Config	Parse, create
31	Detects Suspicious	Fraud score	Fraud Queue	Flag transaction
32	Reviews Flagged	Approval/reject	Payment Release	Release/refund
33	Daily Job Runs	Cron trigger	Analytics Service	Generate reports
34	Session Expires	Inactivity timer	Session Invalidation	Clear token

Logic Breakdown

System-Only Processes (22)

1. **Auto-release locked seats**
   Automatically releases seats after a timeout if checkout is not completed, preventing seat hoarding.

2. **Concurrency resolution**
   Resolves race conditions when multiple users attempt to book the same seat simultaneously.

3. **Inventory synchronization**
   Keeps cached seat availability aligned with the primary database to prevent stale data.

4. **Dynamic pricing adjustment**
   Updates ticket prices based on demand and predefined pricing rules.

5. **Tax calculation**
   Automatically applies region-specific taxes during checkout.

6. **Payment retry logic**
   Retries failed transactions safely without causing duplicate charges.

7. **Email generation**
   Generates booking confirmations, tickets, and cancellation emails automatically.

8. **SMS OTP generation**
   Generates and sends one-time passwords for authentication and verification.

9. **Fraud detection execution**
   Continuously evaluates transactions and behavior to detect suspicious activity.

10. **Report generation**
    Produces sales, booking, and performance reports on a scheduled basis.

11. **Session cleanup**
    Terminates inactive sessions to free system resources and reduce risk.

12. **Cache invalidation**
    Clears outdated cache entries after booking, cancellation, or seat updates.

13. **Recommendation engine execution**
    Generates personalized event recommendations asynchronously.

14. **Waitlist management**
    Activates waitlists when events are sold out and reallocates seats when available.

15. **Event reminder scheduling**
    Schedules reminders based on event date and user preferences.

16. **Refund calculation**
    Computes refund amounts based on cancellation policies and timing.

17. **Data archival**
    Archives historical booking data for compliance and analytics.

18. **Audit logging**
    Records critical system actions for traceability and compliance.

19. **Capacity alerts**
    Triggers alerts when demand approaches venue or system limits.

20. **Session persistence handling**
    Maintains session state across network interruptions.

21. **Rate limiting**
    Restricts excessive requests to prevent abuse and system overload.

22. **Database replication**
    Replicates data across nodes to ensure fault tolerance and availability.

Conditional Branches (8)

1. **Payment method routing**
    Routes transactions to the appropriate gateway based on selected payment method.

2. **Seat availability display logic**
    Determines whether seats are shown as available, locked, waitlisted, or sold out.

3. **User authentication state check**
    Controls access to personalized features based on login status.

4. **Refund eligibility evaluation**
    Applies refund rules based on time remaining before the event.

5. **Error recovery decision logic**
   Determines whether to retry, rollback, or abort an operation after failure.

6. **Accessibility feature activation**
   Enables alternate views or interaction modes when accessibility needs are detected.

7. **Alternate seat suggestion logic**
   Suggests replacement seats when selected seats become unavailable.

8. **Fraud risk threshold evaluation**
   Flags or blocks transactions exceeding predefined risk scores.

Error States (10)

1. **Seat no longer available (SEA_001)**
   Occurs when another user completes booking first.

2. **Insufficient payment funds (PAY_002)**
   Triggered when the payment method lacks sufficient balance.

3. **Invalid or expired card (PAY_003)**
   Raised when card details fail validation.

4. **Session expired (SES_001)**
   Occurs after prolonged user inactivity.

5. **Venue not found (VEN_001)**
   Triggered when a venue is removed or unpublished.

6. **Duplicate booking detected (BKG_001)**
   Prevents repeated bookings for the same user and event.

7. **Invalid OTP entered (SEC_001)**
   Raised when authentication codes do not match.

8. **Payment gateway timeout (GWY_001)**
   Occurs when the payment processor does not respond.

9. **Database connection lost (DB_001)**
   Triggered during backend connectivity failures.

10. **Concurrent booking conflict (CON_001)**
    Occurs during race conditions under high traffic.

Edge and Exception Cases (8)

1. **Last-minute seat conflict**
   High-demand scenarios where many users attempt to book the final seat.
2. **Event cancellation**
   Triggers automated refunds and notifications.
3. **Traffic spike**
   Sudden surge in users requiring rate limiting and load balancing.
4. **Currency fluctuation**
   Exchange rates are locked during checkout to prevent pricing mismatch.
5. **Internet disconnection during payment**
   Requires server-side validation and safe retry handling.
6. **Card expiration during checkout**
   Detected in real time to prevent settlement failure.
7. **Timezone boundary issues**
   All event times are normalized to UTC to avoid inconsistencies.
8. **Accessibility change mid-booking**
   Allows preference updates without restarting the booking flow.

Backend Triggers and Automated Processes (6)

1. **Seat lock timeout trigger**
   Automatically releases locked seats after timeout expiration.
2. **Scheduled inventory synchronization**
   Periodically syncs cache and database inventory.
3. **Dynamic pricing update trigger**
   Adjusts prices based on demand thresholds.
4. **Payment retry trigger**
   Initiates retries after transient payment failures.
5. **Notification dispatch trigger**
   Sends confirmations, reminders, and alerts automatically.
6. **Fraud monitoring trigger**
   Continuously scans transactions and behavior in the background.

https://
amityassignments.pages.dev/
assignment-1/markmap%20(2).svg

3. Missing Feature Identification

This section identifies critical system features that are typically invisible to end users but are essential for reliability, accessibility, scalability, security, and business sustainability. These features are intentionally excluded from the visible functional decomposition to highlight gaps that commonly exist in surface-level product analysis. Each feature is mapped to the same five-layer functional model used in the decomposition tree.

Feature 1: Seat Layout Rotation and Accessibility View

Why it is overlooked: Seat layouts are usually treated as static visual grids with no alternate representations.
Why it is necessary: Accessibility-friendly views are required for wheelchair users, elderly users, and visually impaired users to make informed seating decisions.
Layer: Level 3 – Detailed Functions (Accessibility Views)

Feature 2: Abandoned Cart Recovery Mechanism

Why it is overlooked: Abandoned carts are often considered an analytics concern rather than a functional requirement.
Why it is necessary: A large percentage of bookings are abandoned before payment, and recovery mechanisms significantly improve revenue and seat utilization.
Layer: Level 2 – Sub-Functions (Cart Persistence)

Feature 3: Partial Group Booking (Split Preferences)

Why it is overlooked: Booking flows usually assume a single unified group preference.
Why it is necessary: Groups often have mixed seat preferences, and partial booking prevents loss of the entire transaction.
Layer: Level 4 – Micro-Components (Group Management Logic)

Feature 4: Dynamic Pricing Transparency

Why it is overlooked: Pricing logic is often treated as backend-only computation.
Why it is necessary: Transparency builds user trust and ensures compliance with consumer protection

regulations.
Layer: Level 3 – Detailed Functions (Price Breakdown)

Feature 5: Ticket Transfer and Gifting

Why it is overlooked: Post-booking behavior is frequently deprioritized.
Why it is necessary: Ticket transfers increase platform utility and unlock secondary engagement without resale abuse.
Layer: Level 2 – Sub-Functions (Ticket Transfer)

Feature 6: Event Comparison View

Why it is overlooked: Events are usually presented in isolation.
Why it is necessary: Users benefit from comparing timings, venues, and prices across similar events before committing.
Layer: Level 3 – Detailed Functions (Comparison Tools)

Feature 7: Biometric Authentication

Why it is overlooked: Email and password authentication is considered sufficient by default.
Why it is necessary: Biometric authentication significantly improves security and usability on mobile devices.
Layer: Level 4 – Micro-Components (Biometric Verification)

Feature 8: Real-Time Inventory Redistribution

Why it is overlooked: Venues are treated as independent inventory pools.
Why it is necessary: Redistribution prevents artificial sell-outs when inventory is unevenly allocated across shows or sections.
Layer: Level 1 – Primary Buckets (Quota Redistribution)

Feature 9: Accessibility Compliance Reporting

Why it is overlooked: Compliance is often handled outside the product experience.
Why it is necessary: Regulatory requirements such as WCAG 2.1 demand measurable accessibility compliance.
Layer: Level 5 – System-Only Logic (Compliance Monitoring)

Feature 10: Pre-Booking Accessibility Questionnaire

Why it is overlooked: Immediate booking flows are prioritized over preference collection.
Why it is necessary: Collecting accessibility needs early prevents booking failures and last-minute accommodations.
Layer: Level 3 – Detailed Functions (Preference Collection)

Feature 11: Fraud Detection Dashboard

Why it is overlooked: Fraud systems are typically hidden from operational visibility.
Why it is necessary: Real-time dashboards allow administrators to intervene before financial or reputational damage occurs.
Layer: Level 5 – System-Only Logic (Fraud Monitoring)

Feature 12: Multi-Language and Regional Support

Why it is overlooked: English is often assumed as the default language.
Why it is necessary: Regional language support is critical in multi-lingual markets and directly impacts adoption.
Layer: Level 2 – Sub-Functions (Language Support)

Feature 13: Booking Analytics for Users

Why it is overlooked: Analytics are commonly restricted to administrative use.
Why it is necessary: Providing users with booking history insights improves engagement and retention.
Layer: Level 3 – Detailed Functions (Personal Analytics)

Feature 14: Contingency Seating Plans

Why it is overlooked: Systems assume ideal operational conditions.
Why it is necessary: Equipment failure or venue constraints require rapid reassignment without canceling bookings.
Layer: Level 4 – Micro-Components (Alternate Seating Logic)

Feature 15: Performance Metrics and SLA Monitoring

Why it is overlooked: Performance monitoring is treated as a purely operational concern.
Why it is necessary: Public-facing systems require measurable reliability to maintain user trust.
Layer: Level 5 – System-Only Logic (Status Monitoring)

Feature 16: Subscription and Membership Tiers

Why it is overlooked: Ticket purchases are assumed to be transactional and one-off.
Why it is necessary: Membership tiers enable recurring revenue and preferential booking experiences.
Layer: Level 2 – Sub-Functions (Membership Management)

Feature 17: Corporate and Bulk Booking Portal

Why it is overlooked: Systems are designed primarily for individual users.
Why it is necessary: Corporate bookings represent high-volume, high-value transactions with distinct requirements.
Layer: Level 1 – Primary Buckets (B2B Booking Pipeline)

Feature 18: Historical Analytics and Trend Detection

Why it is overlooked: Systems focus on current availability rather than historical insight.
Why it is necessary: Trend analysis informs strategic pricing, scheduling, and demand forecasting.
Layer: Level 5 – System-Only Logic (Trend Analysis)

Conclusion

This assignment applied functional decomposition to an Event Ticket Booking Application in order to systematically understand its structure, behavior, and underlying logic. By breaking the system into five clearly defined layers, the analysis revealed not only user-visible features but also the hidden system processes that ensure reliability, scalability, and correctness.

The decomposition exercise highlighted the importance of separating core functionality from conditional logic, automation, and exception handling. System-only operations such as seat locking, inventory synchronization, payment verification, and fraud detection play a critical role in maintaining fairness and stability, especially under high concurrency and failure scenarios. Identifying these elements early helps prevent design oversights that are not immediately visible at the interface level.

This structured breakdown will directly support navigation and interaction design in Module II. The clear mapping of user actions to system responses provides a strong foundation for designing intuitive flows, consistent information architecture, and resilient user journeys. Additionally, the identification of missing features and edge cases ensures that future design decisions account for accessibility, error recovery, and real-world usage conditions.

Overall, this functional decomposition demonstrates how complex digital systems can be analyzed methodically, enabling better design decisions, improved user experience, and more robust system behavior in subsequent stages of development.