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.

Navigation Design and Information Architecture

Assignment 2

UX Design Studio – IV

Course Code: UXD 205

Student Name:

Pujan Aggarwal

Student ID:

A021142924039

Submission Date:

22 December 2025

Abstract

This assignment focuses on building a complete navigation architecture for an Event Ticket Booking Application, derived from the functional decomposition created in Assignment 1. The work translates system functionality into structured navigation paths, covering primary, secondary, and tertiary screens along with system-driven and conditional routes. The architecture accounts for user goals, task sequences, and system constraints to ensure clarity and scalability. Special attention is given to error states, authentication flows, and automated system routes to reflect real-world usage conditions.

Problem Definition

Designing a navigation architecture for an Event Ticket Booking Application presents a unique challenge due to the system's functional density and high number of conditional pathways. Unlike simple content-driven applications, this product must support discovery, selection, transaction, and post-booking management while simultaneously handling system-driven states such as seat availability, payment validation, authentication, and error recovery.

The primary problem lies in translating a complex functional structure into a navigation system that remains understandable and usable for end users. Many critical processes such as seat locking, payment retries, refunds, and session expiration are essential to system reliability but are not directly initiated by users. If these system-driven routes are not carefully integrated into the navigation architecture, they can lead to broken flows, confusion, or loss of user trust.

Additionally, users interact with the system in non-linear ways. They may enter the product through different entry points, abandon flows mid-way, encounter unavailable data, or return after long periods of inactivity. The navigation architecture must therefore account for alternate paths, redirection states, and conditional screens without increasing cognitive load.

The core problem addressed in this assignment is to design a navigation structure that balances clarity with completeness. The architecture must expose necessary functionality at the right level, hide system complexity where appropriate, and still provide predictable navigation across normal, exceptional, and failure scenarios.

Summary of Feature Inventory

The feature inventory for this navigation architecture is derived directly from the five-layer functional decomposition developed in Assignment 1. The purpose of this step is to identify all functional elements that require user-facing screens and to distinguish them from system-level logic and background processes. This ensures that the navigation structure is grounded in actual functionality rather than assumed interface elements.

Based on the decomposition, features were classified into three categories: interface-level features, system-level features, and background processes.

Interface-level features are functions that require one or more screens for user interaction. These include event discovery, search and filtering, event detail views, seat selection, cart and checkout, payment selection, ticket access, booking history, cancellations, and profile management. Several of these features require multiple screens, such as the booking flow (event selection → seat selection → order summary → payment → confirmation) and post-booking management (ticket view, cancellation, refund status).

System-level features are logic-driven components that influence navigation but do not always have a dedicated screen. Examples include seat locking, inventory synchronization, dynamic pricing, fraud detection, payment verification, session handling, and refund eligibility checks. While these features operate in the background, they often generate conditional screens such as loading states, validation screens, error messages, or confirmation views that must be represented in the navigation architecture.

Background processes include fully automated operations such as cache invalidation, database synchronization, audit logging, notification scheduling, retry mechanisms, and analytics generation. These processes do not require direct user interaction but trigger system-driven routes like success messages, retry prompts, timeout screens, or forced redirections.

From this inventory, all features requiring a user interface, multiple screens, or dynamic content generation were extracted. These features form the foundation for grouping, navigation clustering, and flow mapping in the subsequent sections. By explicitly separating interface-level functionality from system and background logic, the navigation architecture can remain user-centric while still accounting for real-world system behavior.

Navigation Grouping Logic

Navigation grouping for the Event Ticket Booking Application was derived from the feature inventory identified in the previous section. Features were organized into hierarchical navigation clusters to reduce cognitive load, support task continuity, and align with user goals and system

constraints. The grouping follows a three-level structure: primary categories (Level 1), sub-categories (Level 2), and micro-functions (Level 3).

Level 1: Primary Navigation Categories

The primary navigation consists of high-frequency, goal-oriented sections that represent the core user intents within the system. These include Event Discovery, Bookings, Tickets, Profile, and Support. Each category exists to support a distinct mental model rather than a technical grouping.

Event Discovery groups all features related to browsing, searching, filtering, and comparing events. This category is placed first due to its high entry frequency and its role as the starting point for most user journeys.

Bookings represents active and historical booking-related tasks such as order summaries, cancellations, refunds, and booking status. This category is separated from Tickets to avoid mixing transactional history with access credentials.

Tickets focuses on ticket access, QR codes, transfers, and entry-related actions. This separation ensures that time-sensitive actions are immediately accessible without navigating through booking management screens.

Profile contains user-specific settings including personal details, preferences, saved payment methods, and accessibility options. These features are lower in frequency and are therefore placed away from the primary task flows.

Support includes help resources, FAQs, dispute resolution, and system-generated assistance. This category is intentionally isolated to prevent support actions from interrupting primary flows.

Level 2: Sub-Category Grouping

Within each Level 1 category, features are grouped based on task sequence and functional similarity. For example, Event Discovery contains sub-categories such as Search, Filters, Event Details, and Venue Information. These elements are grouped to support progressive disclosure, allowing users to move from exploration to selection naturally.

In the Bookings category, sub-categories include Active Bookings, Past Bookings, Cancellations, and Refund Status. These groupings reflect different temporal states of a booking rather than feature type.

Tickets include sub-categories such as Ticket Wallet, QR Code View, Ticket Transfer, and Entry Instructions. These are grouped to support fast access during time-critical scenarios such as

venue entry.

Profile sub-categories include Account Details, Preferences, Security, and Payment Methods. Support sub-categories include FAQs, Contact Support, and Issue Tracking.

Level 3: Micro-Function Placement

Micro-functions represent detailed actions that occur within specific screens, such as applying filters, selecting seats, validating OTPs, or confirming payments. These functions are embedded contextually within their parent screens rather than exposed as standalone navigation items. This approach minimizes navigation depth and prevents users from being overwhelmed by excessive options.

Grouping Criteria

All navigation clusters were formed using the following criteria:

- Functional similarity to ensure related tasks are grouped together
- Frequency of access to prioritize commonly used features
- Cognitive load to avoid unnecessary mental switching
- Task sequence to support natural user progression
- User goals rather than system architecture
- System constraints such as validation and state dependency

This grouping logic ensures that the navigation architecture is symmetrical, predictable, and complete, while still flexible enough to accommodate system-driven and conditional flows addressed in later sections.

Global Navigation Strategy

The navigation architecture for the Event Ticket Booking Application adopts a **hybrid navigation strategy**, combining tab-based primary navigation with contextual and system-driven navigation patterns. This approach was selected to balance high-frequency user tasks with complex, conditional flows that occur during booking and post-booking scenarios.

The primary navigation uses a **tab-based structure** to expose the most frequently accessed user goals: Event Discovery, Bookings, Tickets, Profile, and Support. Tab-based navigation is appropriate for this product because users regularly switch between these sections, and the tabs provide persistent visibility and quick access without increasing cognitive load.

Contextual navigation is used extensively within task flows such as booking, seat selection, checkout, and payment. These flows require sequential progression and temporary focus, making them unsuitable for permanent global navigation placement. Contextual screens such as seat maps, order summaries, payment validation, and confirmation pages are therefore accessed through in-flow transitions rather than direct navigation links.

System-driven navigation is used for non-user-initiated transitions, including error states, loading screens, retry prompts, authentication redirects, and session expiration handling. These routes are not exposed in the global navigation but are embedded into the architecture to ensure predictable recovery paths and continuity of user journeys.

Alternative navigation patterns such as drawer-based or fully hierarchical navigation were evaluated and rejected. Drawer-based navigation increases discoverability cost and slows access to time-critical actions such as ticket retrieval. Fully hierarchical navigation introduces unnecessary depth and makes it difficult to handle cross-functional flows such as booking-to-ticket transitions.

The hybrid approach ensures that core user goals remain immediately accessible while complex system behavior is handled contextually, resulting in a navigation structure that is both flexible and scalable.

Complete Navigation Maps

This section presents the complete navigation architecture derived from the feature inventory and grouping logic defined earlier. The diagrams illustrate how primary, secondary, and tertiary screens are connected across the product, including contextual transitions between major task flows. Each map expands a Level 1 navigation category to show its internal structure and relationships.

Conditional and System-Driven Flows

This section documents navigation paths that are triggered by system conditions rather than direct user intent. These include authentication checks, data unavailability, validation failures, and system enforcement routes such as retries, redirects, and forced actions. The following diagrams illustrate how the navigation architecture responds predictably under different conditional states.

Rationale

The navigation architecture presented in this assignment is grounded in the functional decomposition developed in Assignment 1 and refined through feature inventory and grouping analysis. Each Level 1 navigation category exists to represent a distinct user goal rather than a technical system boundary. Event Discovery was prioritized as the primary entry point due to its high frequency of use and its role as the starting point for most user journeys.

Bookings and Tickets were intentionally separated to distinguish between transactional management and time-sensitive access. This separation reduces cognitive load by preventing users from navigating through historical or administrative screens when they need immediate ticket access. Profile and Support were placed as secondary navigation destinations because they are accessed less frequently and typically outside core booking flows.

Screen groupings within each category were designed based on task sequence and state progression. Features that require focused, linear interaction, such as seat selection and payment, were embedded within contextual flows rather than exposed as persistent navigation items. Certain screens were merged or hidden to avoid redundancy, particularly where system-driven logic already enforces progression or validation.

Conditional and system-driven routes were explicitly integrated into the architecture to ensure predictable recovery paths. Error states, retries, authentication redirects, and session handling were treated as first-class navigation elements rather than exceptions. This approach ensures that the navigation structure remains resilient under real-world conditions such as failures, data unavailability, or partial user actions.

Overall, structural decisions were made to balance clarity, scalability, and flexibility while maintaining alignment with user goals and system constraints.

Key Insights and Learnings

This assignment demonstrated that effective navigation architecture cannot be designed independently of system logic. Many critical navigation paths are driven by conditions such as authentication state, availability, validation, and system enforcement rather than direct user intent.

The exercise highlighted the importance of separating interface-level screens from system-driven routes. While users interact with only a subset of the system, the navigation architecture must still account for all possible transitions, including errors, retries, and forced redirects. Ignoring these paths leads to fragile designs that fail under real-world usage.

Another key insight was the value of grounding navigation decisions in functional decomposition rather than assumptions about screens. By deriving navigation directly from system functionality, the architecture remains consistent, complete, and defensible.

Finally, designing navigation as a structured system rather than a collection of pages enables better scalability. As new features or conditions are introduced, they can be integrated into the existing structure without disrupting core user flows.

Wireframing, Prototyping and Interaction Detailing

Course: UX Design Studio – IV
Student Name: Pujan Aggarwal

Wireframes and Interaction Structure

Introduction

The purpose of this stage is to translate the previously developed navigation architecture into structured mid-fidelity wireframes. This stage focuses on layout hierarchy, interaction clarity, task continuity, and logical grouping of interface elements. Visual styling is intentionally minimized in order to prioritise functional decision making.

Screen Inventory

Screen Name	Purpose
Home / Event Discovery	Browse featured and trending events
Search Results	Display filtered event listings
Event Details	Show event description, venue and timings
Seat Selection	Allow users to choose available seats
Seat Unavailable State	Inform user about seat conflict
Add to Cart	Store selected tickets
Order Summary	Display ticket details and pricing
Coupon Apply State	Validate discount logic
Payment Method Selection	Choose payment option
Payment Processing	System loading state
Payment Failure	Retry or change method
Booking Confirmation	Confirm successful booking
Ticket Wallet	Access booked tickets
QR Code View	Enable venue entry

Screen Name	Purpose
Booking History	View past bookings
Cancellation Flow	Initiate refund request
Refund Status	Display refund progress
Login Screen	User authentication
Signup Screen	New user registration
Profile Overview	Manage account details
Edit Profile	Update personal information
Notification State	Booking alerts and reminders
Empty Booking State	No booking available
Loading State	System data fetch
Error State	Generic failure recovery screen

Interaction Logic

Each wireframe includes clearly marked interaction points such as navigation buttons, form inputs, filtering actions, and task progression indicators. Sequential flows are designed to minimise cognitive load and guide users from discovery to confirmation without unnecessary interruptions.

Flow Sequences

Flow 1 – Ticket Booking

Event Discovery → Event Details → Seat Selection → Order Summary → Payment → Confirmation

Flow 2 – Booking Cancellation

Booking History → Booking Details → Cancel Request → Refund Status

Flow 3 – Authentication Requirement

Seat Selection → Login Prompt → Authentication → Resume Booking

Reflection

During wireframing, additional system states such as loading feedback, seat conflict resolution, and retry mechanisms were identified. These screens were incorporated to improve reliability and ensure

continuity of user journeys.

Prototyping and Interaction Design

Prototype Scope

The interactive prototype demonstrates the core experience of discovering an event, selecting seats, completing payment, and accessing tickets. Secondary flows include booking cancellation and profile management. System feedback such as validation errors, loading indicators, and success confirmations are also represented.

Component Library Overview

A consistent interaction system was prepared including reusable navigation bars, card layouts, form fields, button states, and feedback components such as modals and notification banners. This ensures visual and behavioural consistency across screens.

Interaction Blueprint

User actions trigger predictable system responses. Examples include:

- Tap Event Card → Navigate to Event Details
- Select Seat → Lock Seat and Update Cart
- Submit Payment → Validate Input and Process Transaction
- Booking Failure → Display Recovery Options

Conditional navigation paths such as authentication checks and retry flows are embedded to support real-world usage scenarios.

Usability Check Summary

A basic peer review indicated that users hesitated during seat selection due to limited feedback on availability. Improvements were planned by introducing clearer visual indicators and confirmation messages. Payment failure messaging was also refined to provide actionable recovery steps.

Final Reflection

The prototyping stage highlighted the importance of micro-interactions in guiding user confidence. Structured feedback mechanisms improved task clarity and reduced uncertainty during critical transactions.

Interaction Detailing and Flow Completion

Navigation Summary

The prototype includes one complete primary flow (ticket booking) and multiple secondary flows such as profile updates, ticket access, and cancellation management. These flows are interconnected through contextual navigation and system-driven transitions.

Interaction Notes

- Button press states confirm action initiation
- Loading indicators communicate system processing
- Error messages provide retry or alternate options
- Success confirmations reinforce task completion

Interaction Map

The interaction structure supports multiple entry points including direct event discovery, notification prompts, and returning user access through booking history. Transition logic ensures no dead-end navigation paths.

Prototype Screenshots Explanation

Annotated screens demonstrate how users move from event selection to booking confirmation. Alternate decision paths such as payment failure or session expiry are included to represent realistic behaviour patterns.

Reflection

The final prototype iteration improved alignment, spacing consistency, and transition clarity. If extended further, usability testing with a larger user group and high-fidelity visual refinement would enhance overall experience quality.

Prototype Link

Full Interactive Prototype (Mid-Fidelity Screens):

https://www.figma.com/make/HGSS2J6J8911lh7leZCFf8/Untitled?t=5gihM9cCaTZeiuAR-1

https://www.figma.com/make/HGSS2J6J8911lh7leZCFf8/Untitled?fullscreen=1&t=1v9liNx5RgXc6TFO-1&preview-route=%23%2F

https://www.figma.com/design/HWRBPSK3LcaBcj34VgEAu2/Untitled?node-id=0-1&t=0QxxOav36ZvOObMn-1