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.