Software Design Specification

BookForMe: AI-Powered Booking Platform

Department of Computer Science
Final Year Project

Team Members:
Jazib Waqas, Ahmad Hanif, Muhammad Taqi, Taha Hunaid

November 26, 2025

Contents

1 Introduction

1.1 Purpose

The purpose of this Software Design Specification (SDS) is to provide a comprehensive architectural overview of the **BookForMe** platform. It details the system's design decisions, hybrid architectural patterns, data structures, and the workflow logic required to build a robust service booking application.

1.2 Scope

The BookForMe system connects service providers (Vendors) and consumers (Users) in Pakistan through a centralized platform. It solves the inefficiency of manual booking by automating the entire Booking and payment verification process.

The system includes:

- **A User Mobile App:** A complete interface for users to book services, join ranked or casual match lobbies, and interact with the community via forums.
- **An Automated WhatsApp Agent:** An intelligent assistant that handles booking conversations on behalf of the vendor. It manages availability checks, locks time slots to prevent double-booking, and automatically verifies payment receipts using OCR.
- **A Vendor Dashboard:** A management tool for business owners to view their schedule, manage slots, and give final approval on bookings without needing to chat with customers.
- **A Centralized Backend:** A single API layer that keeps data consistent, ensuring that a slot booked via WhatsApp is instantly updated on the App.

1.3 Definitions and Acronyms

To ensure clarity for all stakeholders, the following technical terms are defined:

- **Neuro-Symbolic AI:** A hybrid approach that combines the "Creative" capabilities of a Language Model (to understand speech) with the "Strict" logic of code (to handle payments). This ensures the AI understands the user but cannot "hallucinate" fake bookings.
- **Rich Client (The Mobile App):** A fully-featured application installed on the phone (React Native). It handles complex visuals, maps, and menus directly on the device, offering a smoother experience than a website.
- **Conversational Client (WhatsApp):** A text-based interface where the user interacts via natural language instead of buttons. It relies entirely on the AI Agent to guide the user.
- **LangGraph:** A framework used to build the AI's "brain." It acts like a flowchart manager, ensuring the AI follows specific steps (e.g., Check Availability $\rightarrow$ Ask for Payment) and doesn't get confused.

- **OCC (Optimistic Concurrency Control):** A strategy to prevent double-booking.It "locks" a time slot for a few milliseconds while a booking is being processed to ensure two people don't book it simultaneously.
- **OCR (Optical Character Recognition):** Technology used to "read" text from images. We use this to automatically extract the amount (e.g., "500 PKR") from payment screenshots uploaded by users.

2 System Architecture

2.1 Architectural Design

The system follows a **Service-Oriented Architecture (SOA)** with a clear separation between the Presentation, Business Logic, and Persistence layers.

Uniquely, the system supports two distinct client types:

1. **The Rich Client (Mobile App):** Connects directly via HTTPS/JSON. Best for power users who want to browse categories or join social lobbies.
2. **The Conversational Client (WhatsApp):** Connects via Webhooks to the AI Agent. Best for quick bookings without downloading an app.

Both clients converge on a single backend and database, ensuring a "Single Source of Truth." Dependencies are strictly unidirectional: Clients depend on the Backend, and the Backend depends on the Database.

2.2 System Architecture Diagram

The diagram below illustrates the high-level data flow across the system's three tiers: Client, Backend, and Persistence.

The architecture highlights two distinct communication patterns:

- **Synchronous API Flow (Left):** The **Mobile Application** communicates directly with the **Central Backend API** via JSON/HTTPS requests. This path handles real-time user actions like searching for slots or managing the vendor dashboard.
- **Asynchronous Event Flow (Right):** The **WhatsApp API** operates via Webhooks. When a user sends a message, it triggers an asynchronous event to the backend, which then routes the context to the **AI Receptionist Agent** for processing.

The **Central Backend API** acts as the orchestrator, delegating heavy logic to specialized microservices (AI Agent and OCR Service) while maintaining data consistency in the **Firestore DB**.

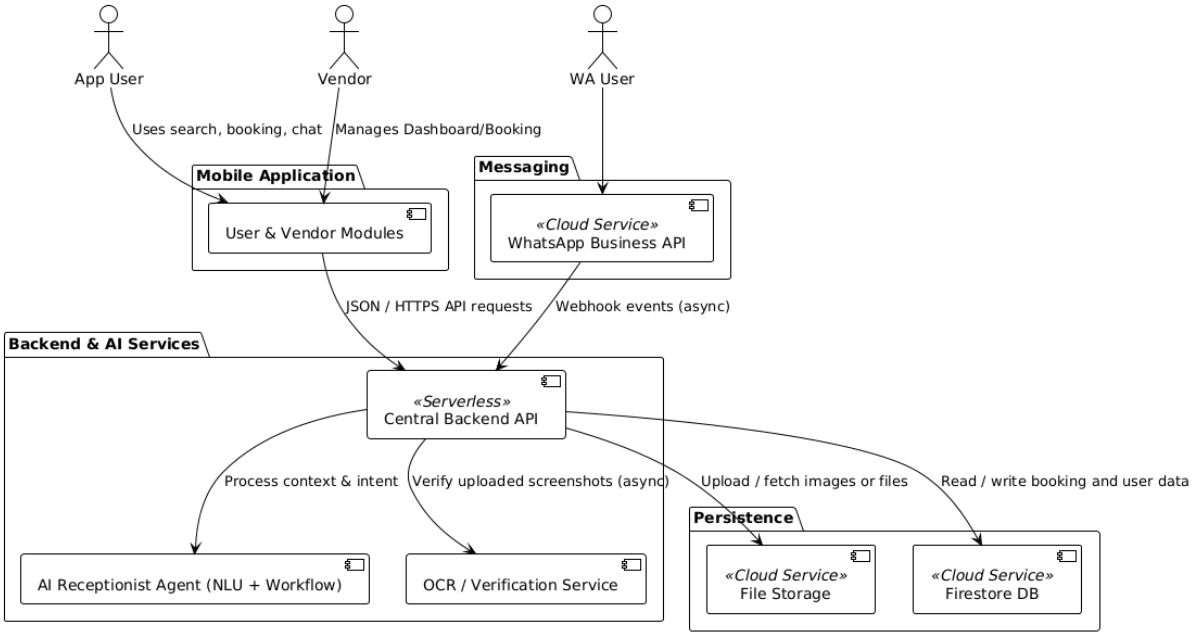

Figure 1: High-Level System Architecture

3 Data Model (Data Design)

3.1 Database Strategy

Data persistence is handled by **Google Cloud Firestore**. We employ a "Reference-Based" schema strategy to balance the flexibility of NoSQL with the need for structured relationships.

The design creates a strict separation between the **Social Ecosystem** (User reputation, Matchmaking) and the **Transactional Domain** (Vendors, Bookings). This ensures that high-volume social interactions (like chatting or posting) do not impact the performance or integrity of critical financial data.

3.2 Entity Descriptions

The table below details the primary entities in the system and their specific responsibilities.

Entity	Key Attributes	Description & Purpose
Users	user_id, auth_uid, elo_rating, skill_profile	The central anchor. Bridges the gap between social features (like skill ratings) and commercial features.
Vendors	vendor_id, location, owner_ref	Represents the business entity. Defined independently of services to allow for scalable inventory management.
Services	service_id, price, type	Represents the sellable unit (e.g., a specific Futsal Court or Gaming PC).
Bookings	booking_id, hold_expires_at, status	The transactional record. Includes a `hold_expires_at` timestamp for the AI Agent's concurrency locking logic.
Payments	transaction_id, ocr_verified_amount, screenshot_url	Separated from bookings to ensure auditability. Stores the raw evidence (screenshot) and the AI-validated amount.
Match_Lobbies	match_id, participants, mode	Manages matchmaking logic, tracking users in Ranked or Casual groups.
Forum_Posts	post_id, content, timestamp	drive user engagement, allowing the community to interact beyond just making bookings.

Table 1: Domain Entity Descriptions

3.3 NoSQL Schema Diagram

The diagram below visualizes the relationships between these entities. The schema is divided into two distinct domains:

1. **User & Social Domain (Left):** Manages user engagement, allowing users to build reputation (Elo) without necessarily making purchases.

2. **Core Booking Domain (Right):** Manages business logic. Note that **Payments** are separated from **Bookings**; this allows the AI to perform OCR verification on the payment receipt before the booking is officially confirmed.

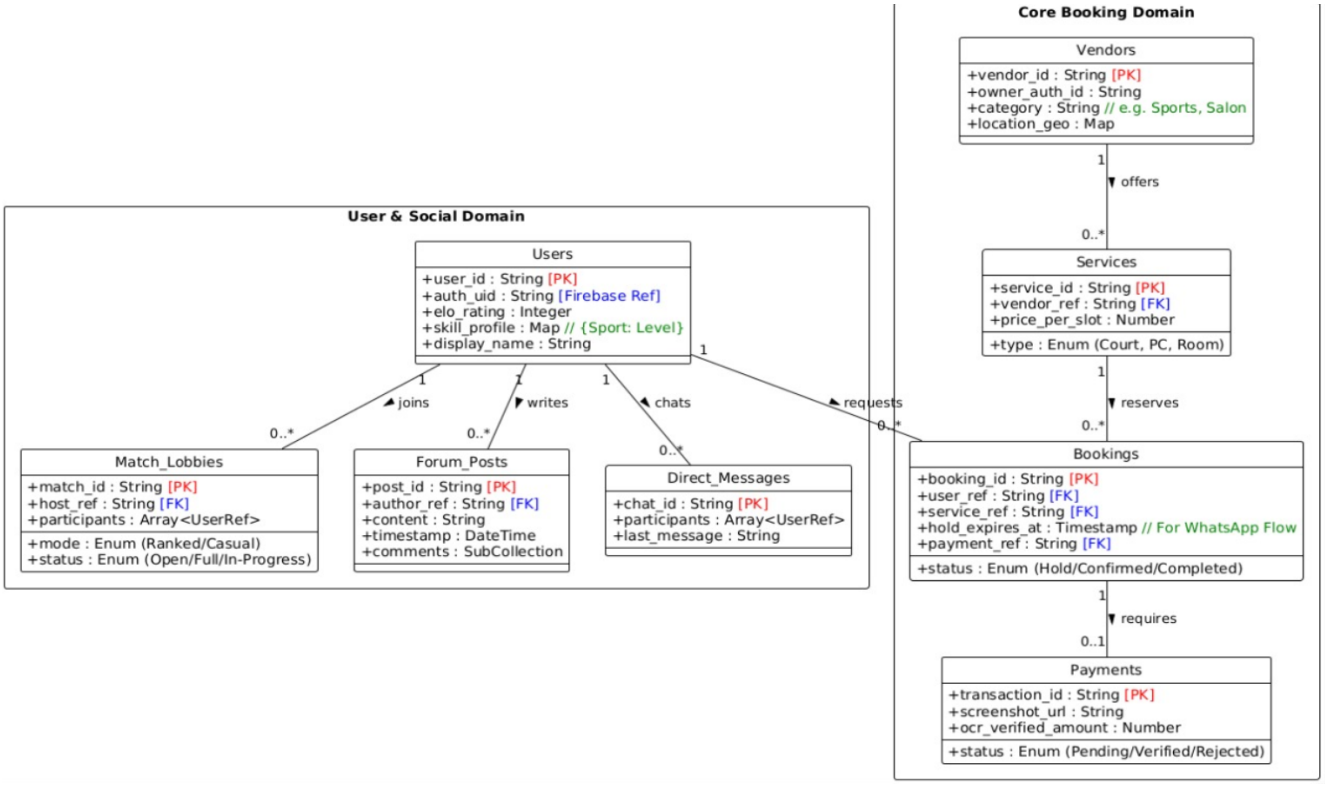

Figure 2: Logical Data Model

4 Agentic Booking Process Pipeline (Sequence Diagram)

Building on the data structures defined in the previous section, this diagram details the specific interaction sequence for the AI-driven booking process. It demonstrates how the system leverages the `hold_expires_at` timestamp and the separated `Payments` entity to manage concurrency and verification in real-time.

Unlike a standard request-response cycle, the AI Agent manages a stateful conversation, enforcing a strict validation pipeline before any data reaches the vendor.

4.1 Pipeline Logic

1. **Ingest & Negotiation:** The Agent maintains a conversational loop with the user to clarify the specific date and time before querying the database.

2. **Transaction (Locking):** Once the intent is finalized, the system utilizes the Optimistic Concurrency Control strategy to create a "HOLD" booking with a 10-minute expiry window.

3. **Intelligent Gatekeeper:** The Agent processes the uploaded payment receipt using the OCR service. If the extracted amount does not match the service price, the system automatically rejects the request. This ensures that the vendor is not disturbed by invalid transactions.

4. **Human-in-the-Loop:** Only validated requests are forwarded to the Vendor Dashboard, where the final approval status is determined.

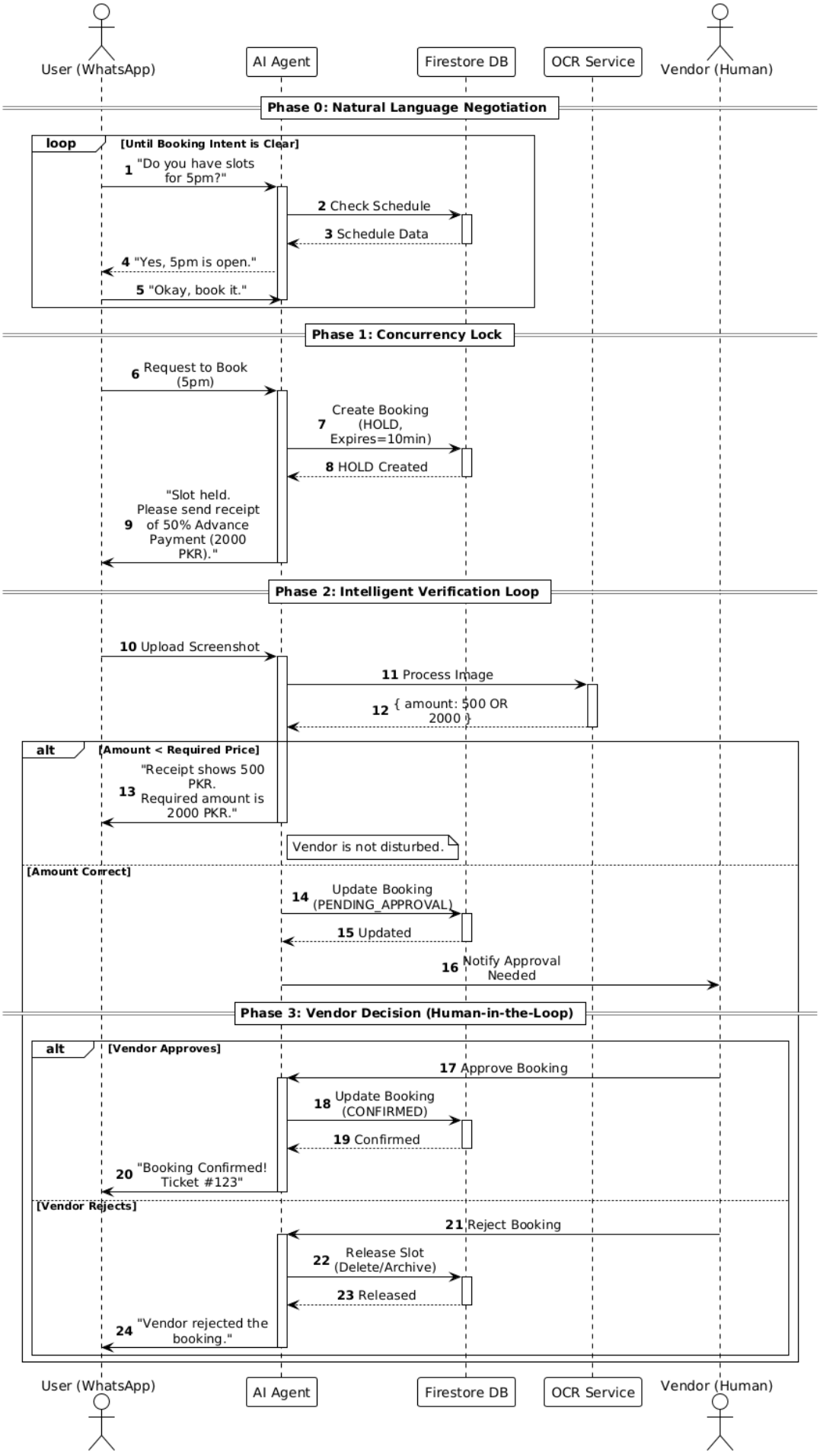

Figure 3: AI Booking Payment Pipeline

5 Technical Details & Workflows

While the previous section illustrated the data pipeline for a single transaction, this section expands on the broader algorithmic logic and user workflows. It details the specific decision trees used to handle complex interactions across the Agent, the App, and the Vendor Dashboard.

5.1 Key Algorithms

The system employs three core algorithms to handle intelligence, concurrency, and social matchmaking.

5.1.1 Algorithm 1: Intent Parsing & Action Routing

Purpose: To translate unstructured natural language into deterministic database actions without hallucination.

- **Input:** Raw WhatsApp Message string + User Session State.
- **Logic:** The LLM classifies the user's intent (e.g., `BOOK_SLOT` vs. `CHECK_PRICE`). The system then "routes" this intent to a specific LangGraph node, which executes rigid code (like checking a specific Firestore document) rather than letting the LLM generate a query itself.
- **Output:** Validated JSON Payload or Error Prompt.

5.1.2 Algorithm 2: Optimistic Concurrency Control (OCC)

Purpose: To prevent race conditions (double bookings) in a distributed NoSQL environment.

- **Input:** `service_id`, `time_slot`, `user_id`.
- **Logic:**
  1. **Read:** Fetch the current slot status.
  2. **Verify:** Check if status is `AVAILABLE`.
  3. **Write:** If valid, update status to `HOLD` with a timestamp. This atomic operation fails if the underlying document changed since the "Read" step.
- **Output:** Transaction Success or Conflict Error.

5.1.3 Algorithm 3: Elo-Based Matchmaking

Purpose: To pair users of similar skill levels for "Ranked" matches.

- **Input:** User's current Elo rating, Preferred Sport, Wait Time.
- **Logic:** The system searches the `Match_Lobbies` collection for open lobbies where the average Elo is within a distinct threshold (e.g., ± 200 points). As wait time increases, the threshold expands to ensure a match is found.
- **Output:** `lobby_id` or `WAIT` signal.

5.2 Workflow Visualizations

To demonstrate how these algorithms manifest in the user experience, the following diagrams detail the specific interaction flows for each primary actor.

5.2.1 Workflow 1: The Agentic Booking Experience

This diagram illustrates the "Conversational Flow" on WhatsApp. It specifically highlights the **Error Correction Loop**: if a user uploads an invalid receipt (detected via OCR), the Agent handles the rejection automatically. The Vendor is only notified once a valid proof is submitted.

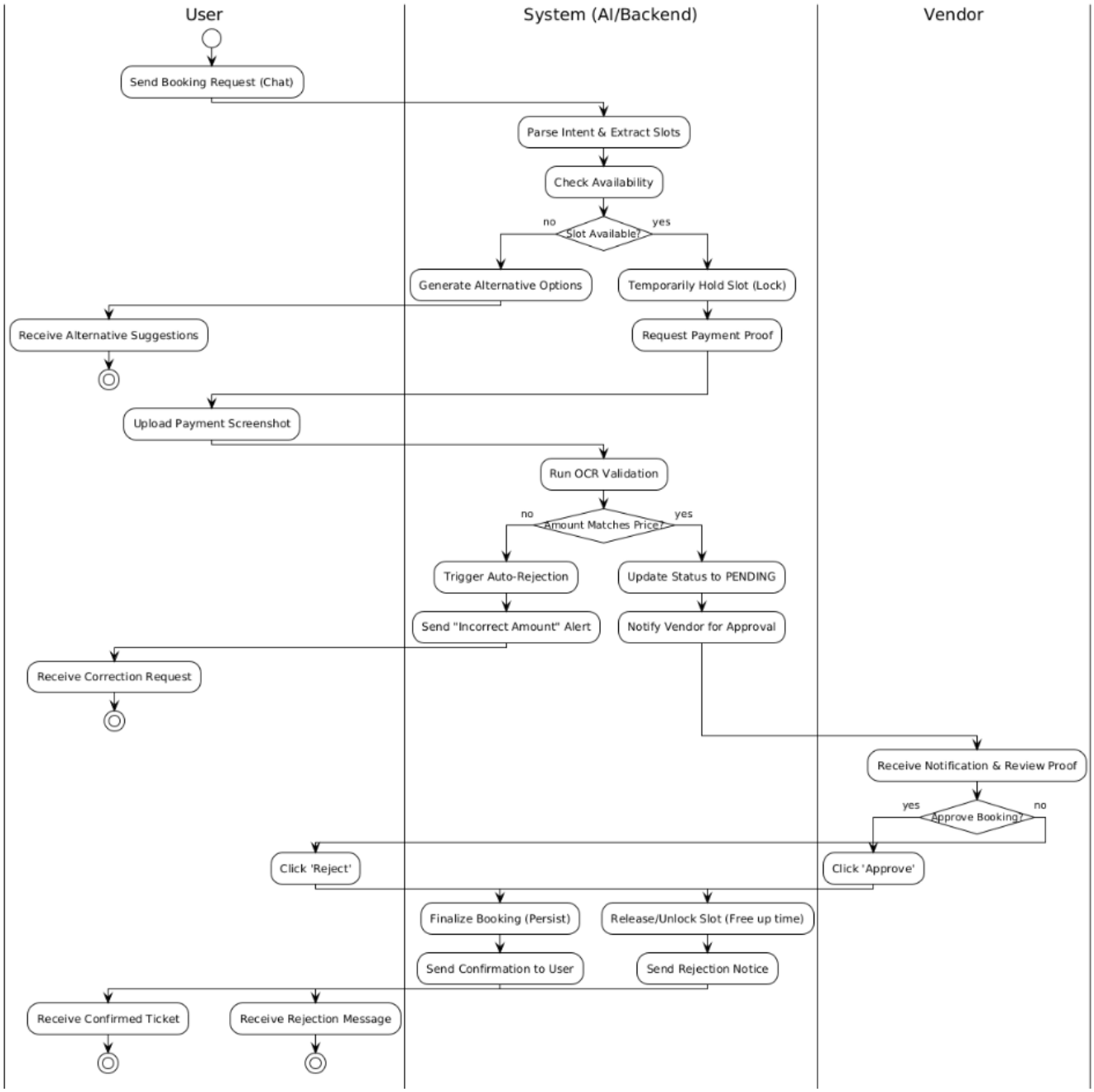

Figure 4: Activity Diagram: Agent/WhatsApp Booking Flow

5.2.2 Workflow 2: The App User Experience

This diagram illustrates the "Rich Client" flow. A key feature highlighted here is the **Dual-Search Capability**: users can browse manually via filters (Location/Price) or use the embedded AI Assistant to find slots via natural language, catering to both power users and casual browsers.

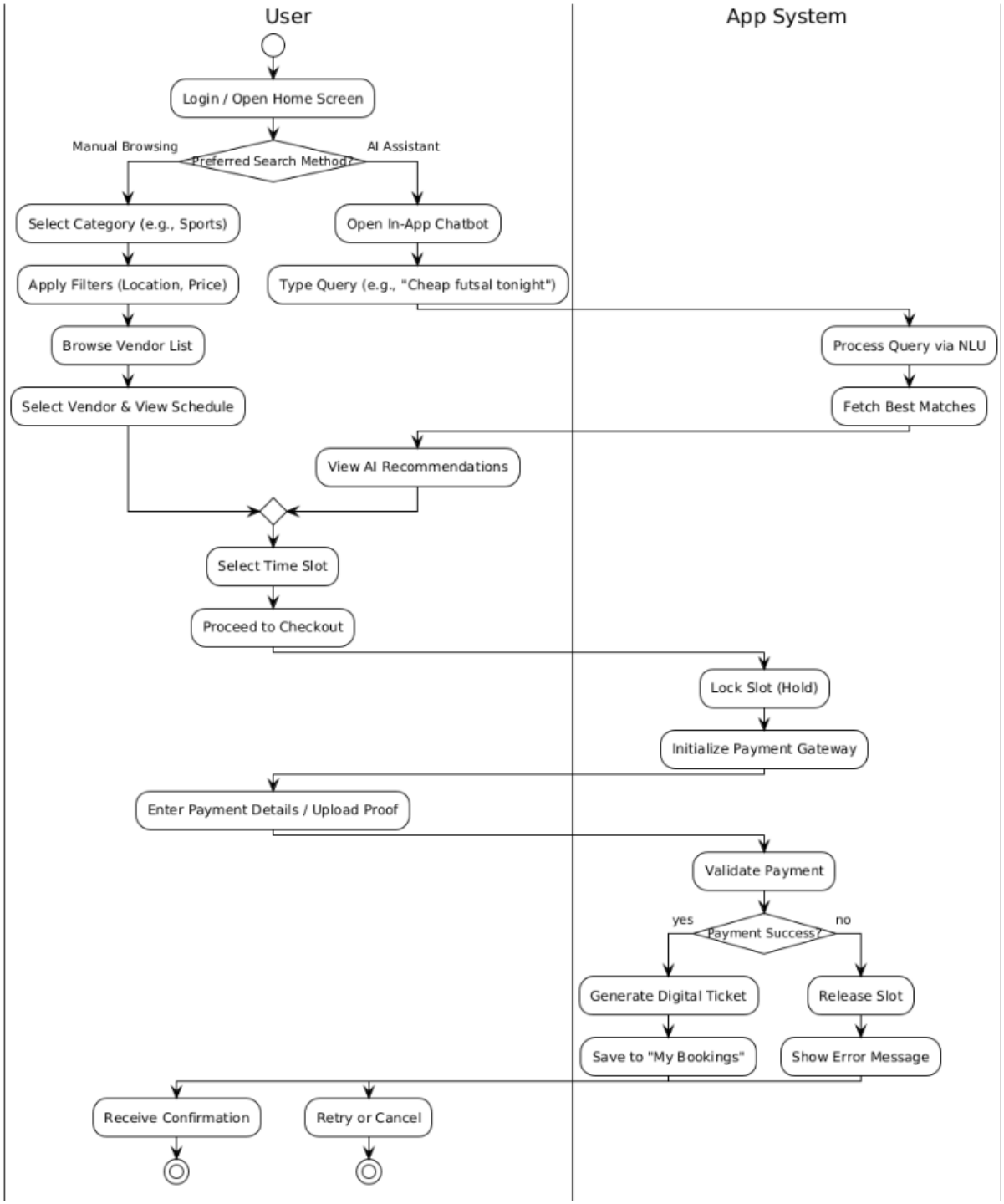

Figure 5: Activity Diagram: Mobile App Search Book

5.2.3 Workflow 3: The Vendor Management Experience

This diagram illustrates the "Human-in-the-Loop" process. It shows how vendors maintain control over their inventory. Crucially, it demonstrates that a **Vendor Rejection** triggers a cascading update in the backend to release the held slot, ensuring the system never deadlocks.

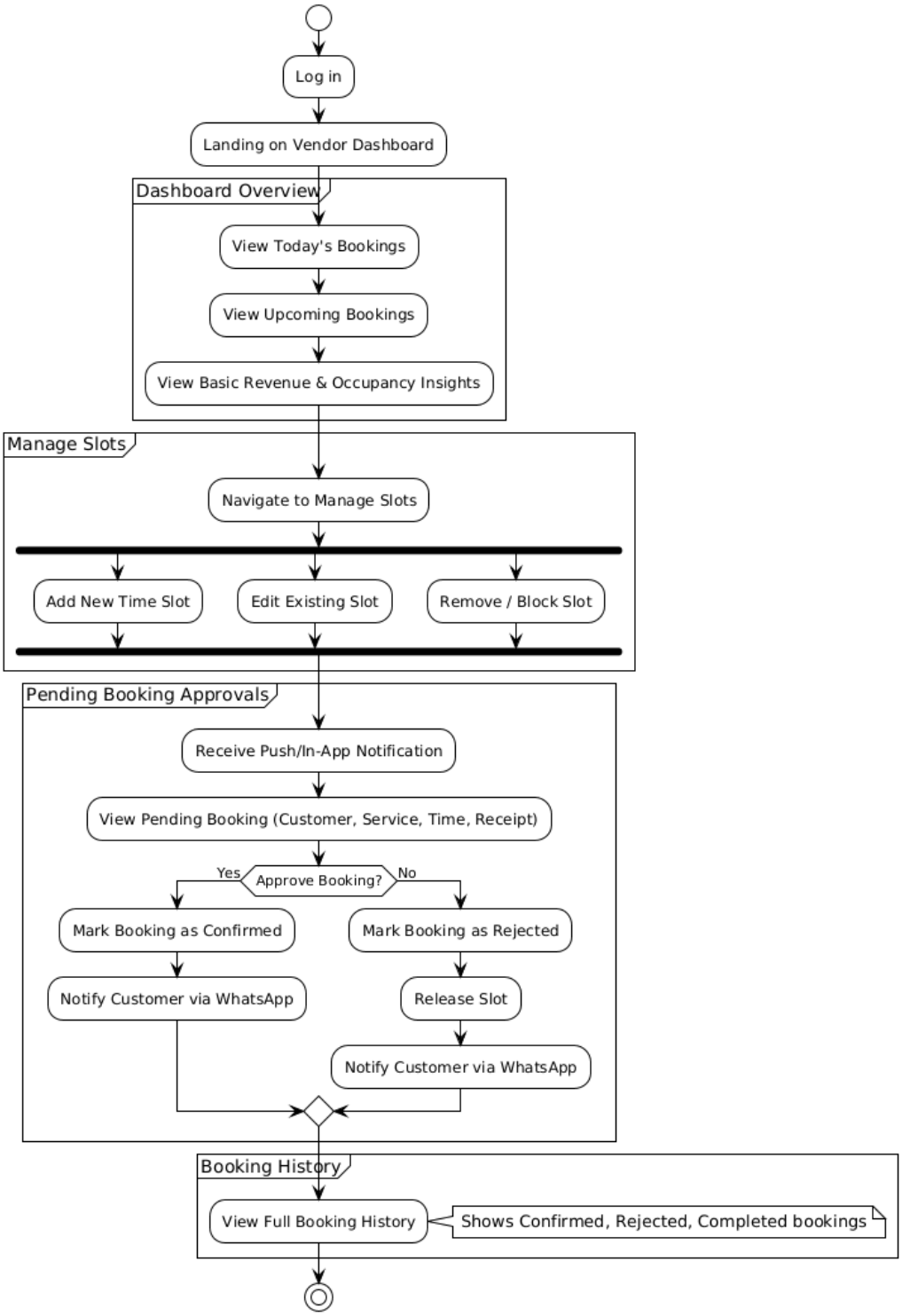

Figure 6: Activity Diagram: Vendor Dashboard Approvals

6 Security Implementation

- **Authentication:** Managed via Firebase Auth. The `auth_uid` in the `Users` collection strictly links the secure token to the application data, preventing unauthorized access.
- **Role-Based Access Control (RBAC):** Custom API middleware validates user claims (Vendor vs. Customer) on every request before granting write access to critical collections like `Services` or `Approvals`.
- **AI Guardrails:** The Neuro-Symbolic architecture acts as a security firewall. Since the LLM cannot execute database queries directly—but must route intents through deterministic code nodes—the system is resilient against "Prompt Injection" attacks where users might try to trick the bot into bypassing payment rules.
- **Data Privacy:** Payment screenshots are stored in a private cloud storage bucket. The system generates time-limited Signed URLs that are only accessible to the specific Vendor and Admin, ensuring financial data remains private.