Product Requirements Document (PRD)

1. Title\ Travel Hotel & Flights Multi-Agent Conversational Planner

2. Purpose & Background\ Provide end users a seamless, stateful chat interface to search and book flights and hotels through natural-language queries. Leverage TypeScript + langgraph.js + gemma3\:latest for intent classification, information extraction, and sub-agent orchestration.\ This PRD will guide implementation in Vibe.Coding using TypeScript.

3. Stakeholders

- Product Owner: Anup Khandelwal
- Engineering Team: Frontend & Backend developers on Vibe.Coding
- QA & Test Engineers
- UX / UI Designers

4. Scope

- **In scope:** Intent detection (flight, hotel, both, other), information extraction (origin, destination, dates, passengers), follow-up dialog, mock search API integration, multi-user session management, streaming responses.
- **Out of scope:** Payment or booking confirmation, real API integration, payment gateways.

5. Functional Requirements

1. **Intent Classification Agent**
2. Input: any natural-language user query.
3. Output: one of { `Flight` , `Hotel` , `Both` , `Other` } via gemma3\:latest.
4. **Entity Extraction Agent**
5. Input: query + detected intent.
6. Output: structured slots (e.g., `fromCity` , `toCity` , `checkIn` , `checkOut` , `departureDate` , `returnDate` , `passengerCount`).
7. Validation: use Zod schemas in TS.
8. **Dialog Manager**
9. Maintains per-user conversation state.
10. Detects missing slots → triggers contextual follow-up questions.
11. Detects completed slot sets → routes to Mock Search Agent.
12. **Mock Search Agent**
13. Implements placeholder functions: `searchFlights()` and `searchHotels()` .
14. Returns simulated results.
15. **Fallback Agent**
16. Handles `Other` intent → replies: *"Sorry, I can't answer that."*

6. Non-Functional Requirements

- **Performance:** 200 ms median intent classification.
- **Scalability:** support 1000 concurrent sessions.

- **Reliability:** 99.9% uptime.
- **Security:** sanitize all user inputs; enforce CORS; no Personal Data Storage.
- **Streaming:** Partial responses streamed as agents finish.

7. System Architecture

```
flowchart TD
  User-->ClassifierAgent
  ClassifierAgent-->DialogManager
  DialogManager-- slots incomplete -->FollowUpAgent
  DialogManager-- slots complete, intent Flight-->FlightSearchAgent
  DialogManager-- slots complete, intent Hotel-->HotelSearchAgent
  DialogManager-- intent Other-->FallbackAgent
  FlightSearchAgent-->ResponseStreamer
  HotelSearchAgent-->ResponseStreamer
  ResponseStreamer-->User
```

8. Agent Workflow & Patterns

- Use langgraph.js Multi-Agent Pattern
- Define sub-agents as nodes with typed I/O.
- Orchestrator node (`DialogManager`) uses LLM calls and schema validation.

9. Tech Stack & Libraries

- **Language:** TypeScript (>=4.x)
- **Agent Framework:** langgraph.js
- **LLM Model:** gemma3\:latest (via Ollama)
- **Schema Validation:** Zod
- **Server Runtime:** Node.js
- **Web Framework:** Fastify (TypeScript) or NestJS
- **State Store:** in-memory Map (for POC); Redis for production.
- **Streaming:** Node.js streams / SSE

10. API & Interfaces

- **REST Endpoints**
- `POST /api/chat`
  - Body: `{ userId: string, message: string }`
  - Response: SSE stream of partial replies.

11. Data Models (Zod Schemas)

```
import { z } from 'zod';

export const IntentSchema = z.enum(['Flight','Hotel','Both','Other']);
```

```
export const FlightSlots = z.object({ fromCity: z.string(), toCity: z.string(),
departureDate: z.string().optional(), returnDate: z.string().optional(),
passengerCount: z.number().min(1) });
export const HotelSlots = z.object({ location: z.string(), checkIn:
z.string().optional(), checkOut: z.string().optional(), guestCount:
z.number().min(1) });
```

Combine for `Both` as intersection.

12. Session & State Management

- Keyed by `userId`.
- Store: `{ intent, slots, stage }`.
- On each message: load state, invoke Classifier → Extraction → Manager.

13. Mock API Design

- `async function searchFlights(slots: FlightSlots): Promise<FlightResult[]>`
- `async function searchHotels(slots: HotelSlots): Promise<HotelResult[]>`

14. Testing Strategy

- **Unit tests:** each agent node.
- **Integration tests:** end-to-end dialogs (using jest + supertest).
- **Mock coverage:** simulate edge cases (vague/incomplete/irrelevant queries).

15. Deliverables

- Complete TypeScript codebase in Vibe.Coding project.
- Automated tests & mock data scripts.
- Clear README.md with architecture, local setup, and run instructions.
- Mermaid diagram in README.

16. Timeline & Milestones

Phase	Duration	Deliverables
PRD & Architecture	1 day	This PRD, Mermaid diagram
Agent Implementation	2 days	Classifier, Extraction, DialogManager
Mock Search Integration	1 day	Flight & Hotel mock agents, sample responses
State & Streaming	1 day	SSE endpoint, session store
Testing & QA	1 day	Unit & integration tests
Final Review	1 day	README, code polish, merge to main branch

This PRD establishes the blueprint for implementing your multi-agent travel planner on Vibe.Coding with TypeScript, langgraph.js, and the gemma3\:latest model.