# Travel Assistant Chatbot — Proof of Concept (PoC) Specification

## Overview

**Problem.** Help a user plan a short city trip via a chat interface that produces a simple, day-by-day itinerary with suggested POIs and hotels.

**Goals.**

* Minimal, server-rendered chat UI (no JavaScript) that accepts natural-language prompts.
* LLM-backed reasoning (OpenAI “GPT-5 Thinking”) orchestrates tool calls to:
  + Find POIs via OpenTripMap.
  + Fetch hotel suggestions via a pluggable “Hotel Provider” wrapper (default: static stub; optional: simple RapidAPI-based provider if key provided).
* Export itinerary as JSON; allow light editing via chat (“swap this museum,” “downgrade budget,” etc.).
* Enforce a monthly LLM spend cap of **$10** with graceful degradation.

**Non-goals.**

* Rich frontend, streaming tokens, maps, or realtime collaboration.
* Flight search, complex pricing/availability guarantees.
* Multi-currency conversions, payment, or booking.

**Success metrics.**

* User can produce a 2–4 day itinerary with at least 6 relevant POIs and 1–3 hotel candidates in ≤ 3 chat turns.
* ≥ 90% of tool calls complete within 2 s excluding LLM latency.
* Spend limiter halts LLM calls when projected monthly spend ≥ $10 and surfaces a clear message.

## Personas & Use Cases

**Personas**

* **Traveler (Guest):** wants quick, sensible suggestions; no account required.
* **Operator (Admin):** checks logs, sets API keys, resets spend cap.

**Top Use Cases**

1. **UC-1: Generate itinerary (baseline).** Traveler enters city, dates, budget tier → gets draft itinerary with POIs + hotels.
2. **UC-2: Refine via chat.** “Replace afternoon museum with a park”; “Prefer budget hotels.”
3. **UC-3: Export.** Download itinerary JSON.
4. **UC-4: Admin ops.** View token usage, costs, and last 50 tool calls.

## Functional Requirements

**FR-1** Chat submit (POST) renders a refreshed page with assistant reply and history.  
**FR-2** System extracts destination, dates, party size (optional), and **budget tier** (“budget”, “mid”, “premium”; EUR/night).  
**FR-3** POI lookup via OpenTripMap wrapper given city coordinates, radius, and categories; results normalized and cached.  
**FR-4** Hotel lookup via HotelProvider interface; default implementation returns deterministic sample results; if RAPIDAPI\_KEY present, call simple aggregator endpoint.  
**FR-5** LLM tool-use: backend detects JSON “actions” emitted by the model and executes tools; merges results back to the model for final answer.  
**FR-6** Spend cap: track tokens and estimated cost per request; when monthly total ≥ $10, block further LLM calls and respond with a capped-mode message and POI/hotel suggestions from deterministic heuristics only.  
**FR-7** Export itinerary as JSON (GET /api/v1/itineraries/{id}) containing days, items, places, and hotels.  
**FR-8** Basic Admin page (password from .env) shows monthly cost, token totals, last errors, and recent tool invocations.  
**FR-9** Input validation: dates (ISO), destination non-empty, tier ∈ {budget, mid, premium}.  
**FR-10** Rate limiting (IP-based): max 30 chat posts/day/IP.

## Non-Functional Requirements

* **Performance.** p95 local API latency (excluding LLM) < 200 ms; OpenTripMap calls < 2 s p95; end-to-end chat response p95 < 12 s.
* **Security.** No PII beyond free-text prompt; Admin protected by HTTP Basic (env user/pass). Secrets only in .env.
* **Reliability.** Graceful degradation to heuristic mode if LLM cap hit or provider failure.
* **Usability.** No JS; forms + server-rendered pages; works on mobile widths ≥ 360 px.
* **A11y.** Semantic HTML, labeled inputs, contrast ratio ≥ 4.5:1.
* **Observability.** Token, cost, and tool-call logs; structured JSON logs.
* **Scalability (PoC).** Single instance; SQLite by default; easy switch to Postgres via DATABASE\_URL.

## Domain Model (textual ERD)

* **Session**(id, created\_at, ip\_hash) 1:N **Message**
* **Itinerary**(id, session\_id, city, country, start\_date, end\_date, budget\_tier, created\_at) 1:N **ItineraryDay** 1:N **ItineraryItem**
* **Place**(id, provider, external\_id, name, lat, lon, categories, rating, address, city, country, raw\_json, last\_synced\_at)
* **Hotel**(id, provider, external\_id, name, lat, lon, price\_eur\_per\_night, rating, address, city, country, url, raw\_json, last\_synced\_at)
* **APICache**(id, provider, endpoint, params\_hash, response\_json, fetched\_at, ttl\_seconds)
* **LLMLedger**(id, session\_id, model, prompt\_tokens, completion\_tokens, cost\_usd, created\_at, month\_key, blocked\_after)

Relationships:

* Session 1:N Message
* Session 1:N Itinerary
* Itinerary 1:N ItineraryDay 1:N ItineraryItem
* ItineraryItem (optional FK) → Place or Hotel

## System Architecture

**Components**

* **Web Backend (FastAPI + Jinja2).**
* **LLM Orchestrator.** Enforces spend cap; formats system & tool prompts; parses tool actions; retries idempotently.
* **OpenTripMap Wrapper.** Thin client (httpx) + response normalizer + cache.
* **Hotel Provider Interface.** HotelProvider ABC; default StaticSampleHotelProvider; optional RapidHotelsProvider.
* **Storage.** SQLite (dev) / Postgres (prod). SQLAlchemy Core + Alembic.

**Data Flow**

1. User submits chat → /chat (POST).
2. Controller saves message → Orchestrator builds LLM request with context & tools.
3. LLM may emit {"action":"search\_pois", ...} or {"action":"search\_hotels", ...}; orchestrator executes wrappers.
4. Orchestrator sends tool results back to LLM; final assistant message saved and rendered.
5. On export, /api/v1/itineraries/{id} returns JSON.

## Interfaces & Contracts

### LLM Tool Action Protocol (model output → backend)

Model is instructed to **only** emit a single JSON object (no prose) when calling a tool:

jsonCopy code{  
 "action": "search\_pois",  
 "args": {  
 "city": "Athens",  
 "lat": 37.9838,  
 "lon": 23.7275,  
 "radius\_m": 4000,  
 "categories": ["cultural", "museums", "parks"],  
 "limit": 12  
 }  
}

Supported action values:

* search\_pois, search\_hotels, finalize\_itinerary, clarify\_missing.

### OpenTripMap Wrapper

* **Endpoint:** GET https://api.opentripmap.com/0.1/en/places/radius
* **Args (subset):** radius, lon, lat, kinds, limit, apikey
* **Normalized Response (example):**

jsonCopy code{  
 "places": [  
 {  
 "provider": "opentripmap",  
 "external\_id": "N12345",  
 "name": "National Archaeological Museum",  
 "lat": 37.989,  
 "lon": 23.733,  
 "categories": ["museums", "cultural"],  
 "rating": 4.7,  
 "city": "Athens",  
 "country": "GR",  
 "address": "28is Oktovriou 44",  
 "raw\_json": { "...": "provider fields" }  
 }  
 ]  
}

* **Caching:** key = SHA256(endpoint|sorted(params)); TTL default 24h.

### Hotel Provider Interface

pythonCopy codeclass HotelProvider(Protocol):  
 def search\_hotels(self, city: str, checkin: str, checkout: str, tier: str, limit: int = 5) -> list[dict]:  
 """Return list of normalized hotel dicts."""

**Normalized Hotel:**

jsonCopy code{  
 "provider": "stub|rapid",  
 "external\_id": "abc123",  
 "name": "Hotel Hermes",  
 "lat": 37.975,  
 "lon": 23.728,  
 "price\_eur\_per\_night": 85.0,  
 "rating": 4.2,  
 "address": "Plaka, Athens",  
 "city": "Athens",  
 "country": "GR",  
 "url": "https://example",  
 "raw\_json": { "...": "provider fields" }  
}

**Default Stub Logic:** deterministic list per city with tier-price mapping:

* budget: 40–100 €/night, mid: 100–200 €/night, premium: 200–400+ €/night.

### REST Endpoints

* GET / → chat page (history, form: destination, dates, budget tier, free-text prompt).
* POST /chat → processes chat, renders template with new assistant message.
* GET /api/v1/itineraries/{id} → JSON export.
* GET /admin (HTTP Basic) → ops dashboard.

**Errors**: 400 validation; 429 rate limit; 503 provider/LLM unavailable; 402 spend cap exceeded (blocked).

## Data Storage (SQL Summary)

**sessions**

* id (PK), created\_at (ts), ip\_hash (text, idx)

**messages**

* id (PK), session\_id (FK), role (enum: user|assistant|system), content (text), tokens\_in (int), tokens\_out (int), cost\_usd (numeric), created\_at (ts)
  + index: session\_id, created\_at

**itineraries**

* id (PK), session\_id (FK), city (text), country (text), start\_date (date), end\_date (date), budget\_tier (text), created\_at (ts)

**itinerary\_days**

* id (PK), itinerary\_id (FK), day\_index (int), date (date)

**itinerary\_items**

* id (PK), day\_id (FK), item\_type (text: poi|hotel|meal|transit), ref\_place\_id (FK nullable), ref\_hotel\_id (FK nullable), start\_time (time nullable), end\_time (time nullable), notes (text)

**places**

* id (PK), provider (text), external\_id (text, unique), name, lat (real), lon (real), categories (jsonb), rating (real), address (text), city (text), country (text), raw\_json (jsonb), last\_synced\_at (ts)

**hotels**

* id (PK), provider (text), external\_id (text), name, lat, lon, price\_eur\_per\_night (numeric), rating (real), address, city, country, url, raw\_json (jsonb), last\_synced\_at (ts)

**api\_cache**

* id (PK), provider (text), endpoint (text), params\_hash (text, unique), response\_json (jsonb), fetched\_at (ts), ttl\_seconds (int)

**llm\_ledger**

* id (PK), session\_id (FK), model (text), prompt\_tokens (int), completion\_tokens (int), cost\_usd (numeric), created\_at (ts), month\_key (text), blocked\_after (bool)

Retention: 30 days for api\_cache & llm\_ledger; 7 days for sessions/messages in PoC.

## UX Flows & Screens (server-rendered, no JS)

**Screen: Chat**

* Components: history (left), reply (right), form: destination (text), start\_date, end\_date, budget tier (select), prompt (textarea), submit.
* Validation errors shown inline above form.
* Assistant replies include: trip summary header, POI list (name, brief), hotels (name, nightly €), and a “Download JSON” link when an itinerary exists.

**Screen: Admin**

* Fields: month, total tokens in/out, cost, cap status, last 50 tool calls (timestamp, action, duration, success/error), last 20 errors.

Empty/error states:

* If spend cap exceeded: banner with message and disable new chat submissions (configurable) or route to heuristic mode.

## Security & Compliance

* **Auth:** Public sessions anonymous; Admin via HTTP Basic (ADMIN\_USER, ADMIN\_PASS).
* **AuthZ:** Admin-only for /admin.
* **Secrets:** .env only; never logged.
* **Data minimization:** No emails/names; IP hashed (SHA256+salt).
* **Logging:** Structured JSON; redact API keys; avoid logging prompts verbatim (hash + length, store content separately if needed).

## Telemetry & Ops

* **Metrics:** requests\_total, request\_latency\_ms, tool\_call\_latency\_ms, openai\_tokens\_in/out, cost\_usd\_total, cache\_hit\_ratio.
* **Logs:** per-request trace\_id, action, outcome, errors.
* **Alerts:** when month spend ≥ 80% cap and at 100% cap.

## Acceptance Criteria (selected)

* **AC-1:** Submitting a valid chat form produces an assistant message that includes at least 4 POIs for major cities (Athens, Paris) using OpenTripMap (live or cached).
* **AC-2:** Changing budget tier affects nightly price ranges in hotel suggestions per tier mapping.
* **AC-3:** When LLM\_MONTHLY\_CAP\_USD budget is exceeded, further LLM calls are blocked and UI shows a clear banner; export still works for existing itineraries.
* **AC-4:** /api/v1/itineraries/{id} returns a schema-valid JSON with days and items.
* **AC-5:** No JavaScript present; Lighthouse reports 0 network JS.
* **AC-6:** Admin page protected; wrong creds → 401.

## Test Strategy

* **Unit:** tool wrappers (OpenTripMap params and normalization), HotelProvider stub, spend calculator, cache key hashing, parser for LLM action JSON.
* **Integration:** chat → orchestrator → tool call → final message; db migrations; admin auth.
* **E2E:** happy-path itinerary for Athens; error-path (provider timeout → graceful degradation).
* **Performance:** simulate 100 sequential chat posts; ensure p95 backend latency targets (excluding LLM).
* **Security:** verify no secrets in logs; admin requires auth; rate limiting enforced.

## Risks & Assumptions

* **R-1** Model may emit malformed JSON. *Mitigation:* tolerant parser + repair attempts; fallback to heuristic mode.
* **R-2** OpenTripMap rate limits. *Mitigation:* caching + backoff; reduced limit.
* **R-3** Hotel API variability. *Mitigation:* stable interface; default stub; feature-flag real provider.
* **R-4** Spend estimate mismatch vs vendor pricing. *Mitigation:* price-per-token configurable; margin factor 1.15.
* **R-5** No JS prevents streaming and UX polish. *Mitigation:* concise responses; pagination via multiple page loads.

**Assumptions**

* Single-region deployment; English only; EUR as currency; SQLite on dev, Postgres in prod.
* OpenAI model name configured as gpt-5-thinking.
* Budget tiers use EUR per **night**.

## Delivery Plan

**Milestone 1 — Scaffold & Storage (2–3 days)**

* FastAPI app, Jinja2 templates, CSS; SQLite; Alembic migrations; entities.
* Exit: runserver boots; / renders; DB migrations up.

**Milestone 2 — OpenTripMap + HotelProvider (2–3 days)**

* Implement wrappers, caching, normalization; stub hotel provider.
* Exit: tool calls return normalized data; cache works.

**Milestone 3 — Orchestrator + Spend Cap (2–3 days)**

* LLM prompts, tool protocol, ledger, cap enforcement.
* Exit: chat produces itinerary; cap toggle tested.

**Milestone 4 — Admin & Export (1–2 days)**

* /admin dashboard, basic auth, /api/v1/itineraries/{id} export.
* Exit: acceptance criteria AC-1..AC-6 pass.

**Definition of Done**

* All ACs met; unit test coverage ≥ 80% core modules; README with setup/run/test; no JS shipped.

## Copilot Execution Guide

**Coding standards.** Python 3.11+, type hints; ruff + black; SQLAlchemy Core; httpx; tenacity for retries; pytest.  
**Guardrails.** No secrets in code; validate inputs; least privilege; redact logs; cap LLM spend.  
**Infra.** Dockerfile; docker-compose for db/app; Alembic migrations; .env template.  
**Docs.** README with env vars; API reference via OpenAPI from FastAPI; example prompts.

**Prompts for copilot.**

* Implement per milestone in order.
* After each feature, run tests and output results; provide code diff.