Agentic AI Travel Advisor — Technical Challenge (Take-Home)

0) Summary

Design and deliver a full-stack travel advisory application that uses **agentic planning** (via **LangGraph**) to orchestrate multiple tools (flights, lodging, events, transit, weather, currency, and RAG). The agent must plan, verify against constraints, repair when needed, and produce a structured itinerary with transparent citations. Include lightweight **auth & multi-tenancy**, basic **production readiness** (health, metrics, rate limits), and a concise developer experience (Docker, migrations, tests). It must use a FastApi Backend, Streamlit python web framework frontend, and a postgresl db (using the sqlalchemy python library).

> **Time expectation:** Aim for a focused, high-quality slice deliverable within ~1 work-week of effort. Implement the core agentic loop, at least **five tools** (one **MCP** tool encouraged), streaming UX, and the must-have auth/prod basics below.

---

1) Goals & User Story

Primary user story: > "As a traveler, I want a 4–7 day itinerary for a destination under a budget, with my preferences (e.g., art museums, toddler-friendly), avoiding overnight flights, and comparing multiple airports or neighborhoods. I expect the system to plan, check constraints, and adjust if something doesn't fit."

Examples of queries the system should handle: - "Plan 5 days in **Kyoto** next month under **$2,500**, prefer art museums, avoid overnight flights, toddler-friendly, compare **KIX** vs **ITM**." - "Make it **$300 cheaper** while keeping 2 museum days." - "If Saturday rains, **swap outdoor** activities for indoor."

Non-goals / out of scope: Real GDS integrations, booking/payments, heavy data scraping, and SSO.

---

2) Functional Requirements (Agentic System)

2.1 LangGraph (Mandatory)

Build an explicit **LangGraph** responsible for planning, tool orchestration, verification, and repair. The graph must: - Use a **typed state** (e.g., Pydantic/TypedDict) carrying: `messages, constraints, plan, working_set, citations, tool_calls, violations, budget_counters, done`. - Implement **conditional edges** (if violations → repair). - Support

parallel branches (e.g., try two airports or two neighborhoods concurrently) and merge with a selector/ranker. - **Checkpoint** after key nodes (e.g., after planning and merges) and recover from invalid model outputs by rolling back to last checkpoint. - Emit **progress events** suitable for streaming to the UI (e.g., "Searching flights (KIX)… Evaluating hotels in Gion… Verifying budget…").

Required nodes (minimum): 1. **Intent & Constraint Extractor** → normalize goal, extract hard/soft constraints (budget, dates, airports, preferences). 2. **Planner** → emits a multi-step plan (ordered tool calls and joins) with dependencies and rough cost/time estimates. 3. **Router/Selector** → chooses which step to execute next; supports parallel execution where applicable. 4. **Tool Executor** → executes calls with timeouts, retries (1 with jitter), caching/dedup by input hash; records timings. 5. **Verifier/Critic** → checks the itinerary/partial plan against constraints (see 2.3) and writes violations into state. 6. **Repair/Re-plan** → mutates choices (airport/hotel/day order) and re-executes only affected steps. 7. **Synthesizer** → fuses tool results + RAG passages; produces itinerary JSON + narrative with citations. 8. **Responder** → streams tokens/progress and assembles final result payload.

State skeleton (illustrative):

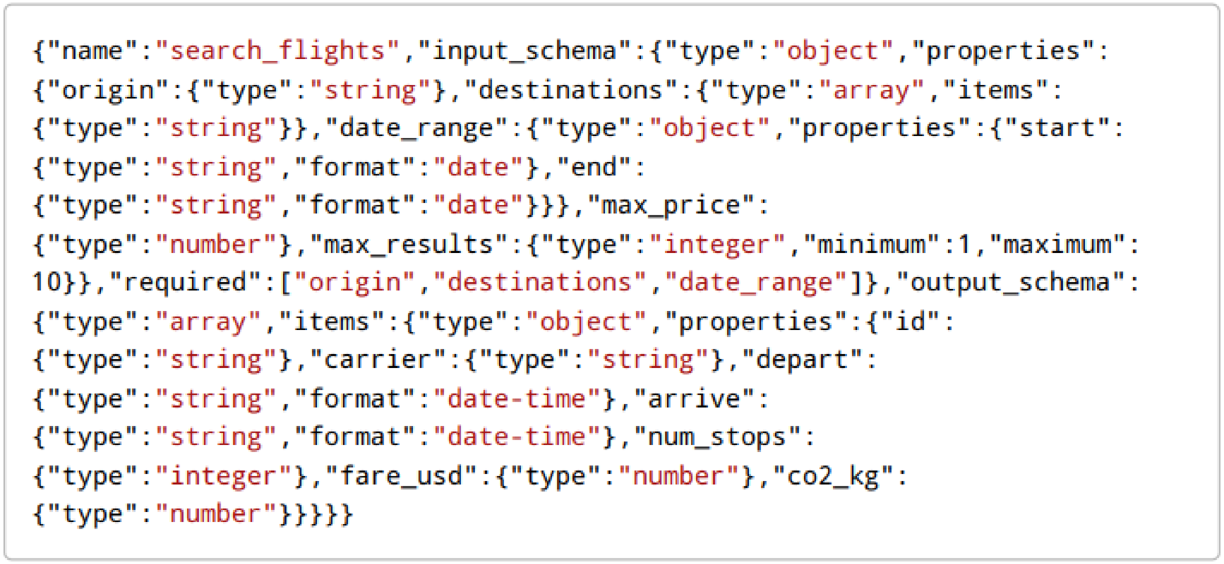
```
{
  "messages": [],
  "constraints": {"budget_usd": 2500, "dates":
{"start":"2025-10-12","end":"2025-10-17"}, "airports":["KIX","ITM"],
"preferences":{"kid_friendly":true,"museums":true}},
  "plan": [{"id":"flights_kix","tool":"flights","args":
{...},"depends_on":[]}, {"id":"flights_itm","tool":"flights","args":
{...},"depends_on":[]}],
  "working_set": {},
  "citations": [],
  "tool_calls": [],
  "violations": [],
  "budget_counters": {"api_cost_est":0.00, "time_ms_used":0},
  "done": false
}
```

2.2 Tools (≥5, multi-domain; at least one via MCP or with MCP-ready adapter)

Implement **at least five** of the following tools with clear JSON input/output schemas. Stubs/fixtures are acceptable where noted. 1) **Flights** — Search round-trip/one-way across date windows & airports; return top N with price & CO_2 estimate (fixture + pricing heuristic acceptable). 2) **Lodging** — Hotels/stays by neighborhood, price, family amenities; include cancellation policy and distance to key POIs. 3) **Events/Attractions** — Museums, concerts, festivals with opening hours and a `kid_friendly` flag (fixture acceptable). 4)

Transit & Travel Time — Door-to-door estimates (train/bus/walk/ride-hail) between hotels and events (fixture or simple graph). 5) **Weather** — Daily/hourly forecast (e.g., Open-Meteo); cache by `(lat, lon, day)`. 6) **Geocoding** — Nominatim or similar; fallback to city center if precise geocoding fails. 7) **Currency Rates** — For budget normalization across sources; daily rates (fixture acceptable). 8) **Knowledge Retrieval** — pgvector-backed retriever over ingested PDFs/Markdown with chunk-level storage.

- **MCP integration (encouraged):** Expose at least **one tool** via an **MCP server** and consume it from the agent. Provide a local fallback implementation with the same schema if MCP is offline.
- **Contracts:** Define JSON Schemas for inputs/outputs. Keep them explicit and small. Example (flights):



```
{"name":"search_flights","input_schema":{"type":"object","properties":
{"origin":{"type":"string"},"destinations":{"type":"array","items":
{"type":"string"}},"date_range":{"type":"object","properties":{"start":
{"type":"string","format":"date"},"end":
{"type":"string","format":"date"}}},"max_price":
{"type":"number"},"max_results":{"type":"integer","minimum":1,"maximum":
10}},"required":["origin","destinations","date_range"]},"output_schema":
{"type":"array","items":{"type":"object","properties":{"id":
{"type":"string"},"carrier":{"type":"string"},"depart":
{"type":"string","format":"date-time"},"arrive":
{"type":"string","format":"date-time"},"num_stops":
{"type":"integer"},"fare_usd":{"type":"number"},"co2_kg":
{"type":"number"}}}}}
```

2.3 Verification & Repair (Decision Quality)

The **Verifier/Critic** must implement at least **four** checks, and the **Repair** node must adjust the plan when violations occur: 1. **Budget** — Sum of flights + hotel + daily spending × days must not exceed `budget_usd`. 2. **Feasibility** — Opening hours align with schedule; transfer buffers ≥ minimum; avoid overnight flights if requested. 3. **Weather sensitivity** — Swap indoor/outdoor days based on forecast; rain/hazard → propose alternatives. 4. **Preference fit** — Respect `kid_friendly`, museum preference, neighborhood safety/quiet preference if modeled.

2.4 RAG (Supportive)

- Maintain a small knowledge base (PDF/MD), chunk (~800–1200 tokens), store chunk text + embedding in **pgvector**.
- Use RAG **for local factual enrichment** (e.g., museum rules, tipping norms) and attach **chunk-level citations** in the final answer.

2.5 UX & Streaming (Functional)

- One conversational thread where a user can:
  - Provide goals/constraints and see **live progress** per node/tool.
  - Receive a structured **itinerary JSON** + Markdown explanation + **citations**.
  - Ask **what-if** refinements (e.g., “Make it $300 cheaper”), triggering the repair path.
- Right rail/footer should show: **tools used** (with durations), **decisions** (e.g., “Chose ITM over KIX due to shorter transfer”), and **constraint checks**.

---

3) Auth & Access (Lightweight, Real)

- **Multi-tenant**: every domain record carries `org_id`; all queries must filter by authenticated user’s `org_id`.
- **Users**: `email` + `password` with **Argon2id** hashes; roles `ADMIN|MEMBER`.
- **JWT (RS256)**: access TTL 15m; refresh TTL 7d with rotation and server-side (hashed) storage by `jti`.
- **RBAC**: `ADMIN` can create users and purge soft-deleted items; `MEMBER` can use all app features within the org.
- **Visibility**: knowledge items support `scope = org_public | private` (private visible only to creator).
- **Security hygiene**: lockout after 5 failed logins (5-minute backoff), input validation on all endpoints, redaction of secrets in logs.

Auth endpoints: `POST /auth/login`, `POST /auth/refresh`, `POST /auth/logout`, `GET /auth/me`, `POST /users` (ADMIN).

---

4) Production Readiness (Pragmatic)

- **Health**: `GET /healthz` checks DB connectivity, presence of embeddings table, and one outbound tool (HEAD with 1s timeout).
- **SLOs (p95, local/dev):** CRUD < 300ms; agent (fixtures) < 5s; agent (real tools) < 12s.
- **Rate limiting**: token bucket — CRUD **60/min/user**, agent **5/min/user**; return 429 + `Retry-After`.
- **Observability**: JSON logs with `trace_id` and `user_id`; per-node timings; cache hit rate; tool error counts; `GET /metrics` (Prometheus or JSON).
- **Idempotency**: write endpoints accept `Idempotency-Key` header. Store short-TTL keys.
- **CORS**: allow only your Streamlit origin; set standard security headers.

---

5) Data Model (Illustrative Excerpts)

Core tables: `org`, `user`, `refresh_token`, `destination`, `knowledge_item`, `embedding` (`pgvector`), `agent_run` (audit/tool log), optional `itinerary` for persistence.

Embedding table (example):

```sql
CREATE EXTENSION IF NOT EXISTS vector;
CREATE TABLE embedding (
  id BIGSERIAL PRIMARY KEY,
  knowledge_item_id BIGINT NOT NULL REFERENCES knowledge_item(id) ON DELETE
CASCADE,
  chunk_idx INT NOT NULL,
  content TEXT NOT NULL,
  embedding VECTOR(1536) NOT NULL,
  created_at TIMESTAMPTZ DEFAULT now()
);
CREATE INDEX embedding_ivfflat ON embedding USING ivfflat (embedding
vector_cosine_ops) WITH (lists = 100);
```

Access control fields: add `org_id` and `created_by` to domain tables; queries must filter by `org_id`.

Agent audit:

```sql
CREATE TABLE agent_run (
  id BIGSERIAL PRIMARY KEY,
  org_id BIGINT NOT NULL REFERENCES org(id) ON DELETE CASCADE,
  user_id BIGINT NOT NULL REFERENCES "user"(id) ON DELETE SET NULL,
  started_at TIMESTAMPTZ DEFAULT now(),
  finished_at TIMESTAMPTZ,
  status TEXT,
  plan_snapshot JSONB,
  tool_log JSONB,
  cost_usd NUMERIC(8,2),
  trace_id TEXT
);
```

6) API Surface

- **Agent**: `POST /qa/plan` (non-streaming) → itinerary JSON + citations + tool log; `WS /qa/stream` or SSE for progress + final payload.
- **Destinations**: `GET/POST/PUT/DELETE /destinations` (soft delete; keyset pagination; idempotent writes).
- **Knowledge**: `GET/POST/PUT /knowledge`, `POST /knowledge/{id}/ingest-file` (PDF/MD → text → chunks → embeddings).

- **Auth**: `POST /auth/login`, `POST /auth/refresh`, `POST /auth/logout`, `GET /auth/me`, `POST /users` (ADMIN).
- **Ops**: `GET /metrics`, `GET /healthz`.

Model output contract (validate via Pydantic):

```
{
  "answer_markdown": "...",
  "itinerary": { "days": [ { "date": "", "items": [
{"start":"","end":"","title":"","location":"","notes":""} ] } ],
"total_cost_usd": 0 },
  "citations": [ { "title": "", "source": "url|manual|file|tool", "ref":
"knowledge_id or tool_name#id" } ],
  "tools_used": [ { "name": "", "count": 0, "total_ms": 0 } ],
  "decisions": [ "Chose ITM over KIX due to shorter transfer time" ]
}
```

---

7) Frontend (Streamlit)

- **Pages**
  - **Destinations**: search, tag filters, add/edit, soft delete; show last agent run for the selection.
  - **Knowledge Base**: list with version history; upload PDF/MD; ingestion progress; preview chunks.
  - **Plan**: chat-like interface with streaming; right rail shows **tools used**, **decisions**, **constraint checks**, **citations**.
- **Streaming**: WS/SSE; render progress updates per node/tool; final structured payload + Markdown summary.

---

8) Developer Experience & Delivery

- **Repo layout**: `frontend/`, `backend/`, `infrastructure/`.
- **Containerization**: one **Dockerfile** per service; **Docker Compose** with Postgres (+pgvector), Redis (cache/rate-limit), optional MCP server.
- **Migrations**: Alembic; seed script creates 3 destinations + several knowledge items.
- **Deps**: pinned via `pip-tools` or `poetry.lock`.
- **CI (minimal)**: lint (ruff/black), mypy, unit tests, build images.
- **Config**: `.env.example` with API keys, JWT keys, origins, rate limits.
- **Security**: no secrets committed; HTST/headers via proxy or app.

---

9) Evaluation & Rubric (100 pts)

- **Agentic behavior (30)** — Clear plan; parallel branches; verification & repair loop; termination criteria; checkpoints.
- **Tool integration (25)** — ≥5 tools; at least one via MCP or MCP-ready; schemas; caching; retries; graceful fallbacks.
- **Verification quality (15)** — Budget/feasibility/weather/preferences checks implemented and effective.
- **Synthesis & citations (10)** — Coherent itinerary; transparent citations for RAG/tool claims.
- **UX & streaming (10)** — Progress visibility; what-if replanning; readable final output.
- **Ops basics (5)** — Health, metrics, rate limits, idempotency.
- **Auth & access (5)** — JWT, roles, org scoping; basic lockout and validation.
- **Docs & tests (5)** — Setup clarity; graph diagram; scenario suite; a few unit/integration tests.

Partial credit is awarded: if time is tight, implement the full agent loop for one destination and two airports with fixtures, and document trade-offs.

---

10) Getting Started

1. **Clone & configure**: copy `.env.example` → `.env`; generate RSA keys for JWT; set Streamlit origin & rate limits.
2. **Run**: `docker compose up` (FastAPI, Streamlit, Postgres, Redis, optional MCP server).
3. **Seed**: run seeding to add sample destinations and knowledge items.
4. **Try**: open the UI, select a destination, ingest a PDF/MD guide, and ask the Kyoto scenario.
5. **Switch**: toggle between fixtures and real tool APIs in config.

---

11) Test & Scenario Suite

Provide a tiny **YAML scenario suite** (10–12 cases): each with `input`, `must_call_tools`, `must_satisfy` (budget, no-overnight, kid-friendly), and expected fields. Include a CLI: `python -m eval/run_scenarios` → prints pass/fail per rule and per node timings.

Unit tests (illustrative): - Auth: login/refresh/logout; refresh rotation & revocation; lockout timing. - Access control: org scoping on CRUD and `/qa/*`. - Rate-limit: exceeding agent calls yields 429 with `Retry-After`. - Health: `/healthz` fails when DB down or tool headcheck fails. - Agent: planner produces parallel branches; verifier detects an overnight flight; repair swaps to a valid option.

12) Submission

- A link to a **public Git repo** with instructions to run locally via Docker Compose.
- A short **screen capture (2–3 min)** demonstrating: parallel airport comparison → verifier flags violation → repair → final itinerary → what-if adjustment.
- Optional: published image(s) to a registry with a one-liner `docker compose pull && docker compose up`.

13) Candidate Checklist (Quick)-make sure everything here is done.

- Mono-repo with a FastApi Backend, Streamlit python web framework frontend, and a postgresl db (using the sqlalchemy python library)
- LangGraph with typed state, conditionals, **parallelism**, checkpoints.
- ≥5 tools; **1 via MCP** (or MCP-ready) with fallback; schemas defined.
- Verifier checks **budget, feasibility, weather, preferences**; repair loop modifies plan.
- RAG with pgvector; **chunk-level citations** in final answer.
- Streaming progress + final structured payload in UI.
- JWT auth, roles, org scoping; lockout & validation.
- Health, metrics, rate limit; idempotent writes.
- Docker Compose, Alembic, seed, pinned deps; minimal CI.
- Scenario suite & a handful of unit/integration tests.
- Brief README with graph diagram and trade-offs.

14) Appendix (Helpful Artifacts)

A. Tool registry format (example)

```
{
  "flights": {"type": "http|mcp", "endpoint": "http://…", "timeout_ms": 2000,
"retry": 1},
  "lodging": {"type": "fixture", "path": "./fixtures/lodging.json"},
  "events": {"type": "mcp", "server": "mcp://events?token=…"},
  "transit": {"type": "fixture", "path": "./fixtures/transit.json"},
  "weather": {"type": "http", "endpoint": "https://api.open-meteo.com"},
  "currency": {"type": "fixture", "path": "./fixtures/rates.json"},
  "geocoding": {"type": "http", "endpoint": "https://
nominatim.openstreetmap.org"},
  "retriever": {"type": "local", "db": "postgresql://…"}
}
```

B. Progress/telemetry event (example)

```
{"trace_id":"…","node":"flights_kix","status":"running","ts":"…","args_digest
":"…"}
```

C. Minimal security headers

```
X-Content-Type-Options: nosniff
Referrer-Policy: same-origin
Content-Security-Policy: default-src 'self'; connect-src 'self' https://your-
streamlit-origin; img-src 'self' data:
Strict-Transport-Security: max-age=31536000; includeSubDomains
```

D. ASCII Graph (illustrative)

```
[Intent] → [Planner] → ┬→ [Flights(KIX)]
                       ├→ [Flights(ITM)]
                       └→ [Hotels(N1,N2)]
            (merge) → [Selector] → [Transit/Weather/Events] → [Verifier]
                                             ↑          |
                                             └──< [Repair ] <── violations

                              → [Synthesizer] → [Responder]
```