Travel Management Platform – Technical Documentation

System Overview

The platform delivers a role-based travel workflow consisting of: - FastAPI backend (`src/main.py`) exposing authentication, employee, manager, and HR AI routes. - Vue 3 front-end (`frontend/vue-project`) for employees, managers, and HR desks. - PostgreSQL database accessed through utility functions in `src/db`. - Retrieval augmented generation (RAG) policy assistant backed by Milvus vector search. - Model Context Protocol (MCP) airline and hotel servers orchestrated through the HR chat endpoint.

High-Level Architecture

```
[Vue 3 SPA] --REST--> [FastAPI Backend] --SQL--> [PostgreSQL]
                                     \--gRPC--> [Milvus Vector DB]
                                     \--MCP--> [Airline MCP Server]
                                     \--MCP--> [Hotel MCP Server]
                                     \--HTTP--> [Azure OpenAI]
```

- Employees interact with the SPA, which calls FastAPI via Axios.
- FastAPI performs JWT auth, validates business rules, and interacts with Postgres.
- The policy assistant embeds queries, searches Milvus, and invokes Azure OpenAI (via LangChain LiteLLM).
- HR chat binds MCP tools to the LLM for tool-augmented completions.

Backend Components

Entry Point (`src/main.py`)

- Configures CORS, mounts routers, and defines `/` and `/health` endpoints.
- Exposes OpenAPI metadata via `FastAPI(title="Travel Management System", version="2.0.0")`.

Authentication (`src/auth/jwt_service.py`)

- Supports OAuth2 password grant via `/auth/token`.
- Issues and revokes JWTs with configurable TTL (`ACCESS_TOKEN_EXPIRE_MINUTES`).
- Maintains an in-memory token blacklist; replace with Redis for production.
- Performs credential lookup against the `users` table (email or employee ID).

Routers (`src/api`)

- `auth_router.py`: register, login, logout, profile endpoints.

- `employee_router.py`: indent CRUD, policy chat, frequent routes, and profile retrieval.
- `manager_router.py`: queue views, approvals, and employee profile lookups.
- `hr_mcp_router.py`: AI chat, ticket status updates, session lifecycle, MCP health.

Services (`src/api/services`)

- `session_service.py`: in-memory session management used by HR chat.
- `travel_indent_service.py`: data access layer for HR functions (status updates, fetch by ID).
- `context_service.py`: enriches HR chat prompts with indent context.

Data Access Layer (`src/db`)

- `connection.py`: connection pooling helper (psycopg).
- `travel_queries.py`: bulk of SQL for indents, approvals, bookmarks.
- Ensures duplicate detection on active trips and enforces manager approval prerequisites.

RAG Policy Assistant (`src/rag`)

- `embedder.py`: Azure OpenAI embeddings configuration.
- `milvus_store.py`: Milvus collection bootstrap, insert, and vector search.
- `rag_service.py`: `PolicyRAGChatService` combining retrieval, context building, and LLM calls.

LLM and MCP Integration

- `src/config/llm_config.py`: constructs `ChatLiteLLM` using Azure deployment.
- `src/api/handlers/mcp_handler.py`: lazy initializes `MultiServerMCPClient`, binds tools to the LLM, and exposes availability metrics.
- `src/config/mcp_config.py`: builds server command definitions from environment variables, enabling stdio MCP transport.

Agents (`src/agents`)

Legacy synchronous agents (employee, manager, HR) remain for CLI usage and illustrate business rules. The HR agent automates sequential booking logic by calling database helpers.

Frontend Overview (`frontend/vue-project`)

- Vite-powered Vue 3 SPA with Pinia state management and Vue Router.

- `src/views/DashboardLayout.vue`: shell that switches between employee, manager, and HR dashboards.
- `src/views/dashboards/EmployeeDashboard.vue`: main employee experience with indent form, frequent route component, policy assistant drawer, and status lanes.
- `src/components/FrequentRoutes.vue`, `MetricCard.vue`, `TravelList.vue`: reusable UI primitives.
- `src/stores/auth.js`: persists JWT, refreshes profile, and handles logout across browser sessions.
- `src/services/api.js`: Axios instance injecting bearer tokens and handling 401 fallbacks.

Data Model Snapshot

Key tables referenced in the codebase: - `users`: employee directory with `employee_id`, `role`, `grade`, `department`, and hashed passwords. - `travel_indents`: primary travel request record including status fields (`is_approved`, manager and HR markers). - `employee_route_bookmarks`: frequent route storage with usage metadata. - `tied_up_hotels`: catalog used to filter eligible hotels by city/grade (queried via `fetch_eligible_hotels`). - `approval_workflow`: tracks approval events by persona.

Environment Configuration (`src/config/settings.py`)

Settings load from `.env` via python-dotenv. Notable keys: - `AZURE_API_KEY`, `AZURE_API_BASE`, `AZURE_API_VERSION`, `AZURE_MODEL`. - `DB_HOST`, `DB_NAME`, `DB_USER`, `DB_PASSWORD`, `DB_SSLMODE`. - `SECRET_KEY`, `ALGORITHM`, `ACCESS_TOKEN_EXPIRE_MINUTES`. - `MILVUS_URI`, `MILVUS_TOKEN`, `MILVUS_COLLECTION_NAME`, `MILVUS_DIM`. - `MCP_AIRLINE_COMMAND`, `MCP_AIRLINE_ARGS`, `MCP_HOTEL_COMMAND`, `MCP_HOTEL_ARGS`. - `SESSION_TIMEOUT_HOURS` controlling both HR chat and policy assistant sessions.

Deployment and Operations

Local Development

1. `python3.11 -m venv .venv && source .venv/bin/activate`.
2. `pip install -r requirements.txt`.
3. Configure `.env` with database, Azure OpenAI, and Milvus credentials.
4. Run FastAPI: `uvicorn src.main:app --reload --port 8000`.
5. Launch Vue UI: `cd frontend/vue-project && npm install && npm run dev`.
6. Optional: start MCP servers (`python src/mcp_servers/airlines/airline_booking_server.py`, etc.).
7. Seed or migrate the PostgreSQL schema via scripts in `scripts/`.

Containerisation

- `Dockerfile` defines the FastAPI service image; `docker-compose.yml` can orchestrate API, database, Milvus, and MCP services.
- Ensure network access between containers and provide `.env` overrides via compose secrets.

Observability

- Use `/health` and `/hr-mcp/health` for liveness diagnostics.
- Extend logging by integrating Python's `logging` module in routers and services (currently minimal print statements).
- Introduce structured logs and request tracing before production.