AltairGo Intelligence — Complete Project Deep Dive

> ***One-liner***: *An AI-powered travel planning platform that uses Google Gemini LLM to generate hyper-personalized, budget-accurate, day-by-day itineraries — with real images, affiliate monetization, and an admin control plane.*

1. Project Identity

Field	Value
Name	AltairGo Intelligence
Repo	`yash-dev007/AltairGo-Intelligence`
License	MIT
Status	In Active Development
Deployment Target	Vercel (frontend SPA) + Flask server (backend)

2. Tech Stack

Frontend

Layer	Technology
Framework	**React 19** (Vite 7)
Routing	React Router DOM v7
Styling	CSS Modules with Glassmorphism aesthetics
Animation	Framer Motion
Maps	Leaflet + React-Leaflet
Drag & Drop	@dnd-kit (sortable activities)
Icons	Lucide React
State Mgmt	React Context API (`AuthContext`)
API Layer	Native `fetch` via `TripAI.js` service

Backend

Layer	Technology
Framework	**Python Flask**
AI Engine	Google Gemini API (via REST HTTP, `gemini-2.0-flash`)
Database	**SQLite** (dev) / PostgreSQL (prod) via **SQLAlchemy ORM**
Auth	**Flask-JWT-Extended** (7-day tokens, `werkzeug` password hashing)
Validation	**Pydantic** schemas + custom `ItineraryValidator`
Image Fetching	5-source priority chain (Cache → Wikipedia → Wikidata → Pexels → SVG Placeholder)
Geo Services	OpenStreetMap Nominatim + Overpass API

DevOps / Config

Tool	Purpose
Vite	Dev server with smart proxy (HTML bypass for SPA routes)
Vercel	Frontend deployment (`vercel.json` SPA rewrites)
Gunicorn	Production backend server
dotenv	Environment variable management
ESLint	Code quality

3. Architecture Overview

DIAGRAM (MERMAID NOTATION)

```
graph TB
    subgraph Frontend["React 19 Frontend (Vite)"]
        App["App.jsx — Router + Layout"]
        Pages["20+ Page Components"]
        Components["16 Component Groups"]
        TripAI["TripAI.js — API Client"]
        Auth["AuthContext — JWT State"]
    end

    subgraph Backend["Flask Backend"]
        AppPy["app.py — Factory + Blueprint Registry"]
        subgraph Routes["8 Blueprints"]
            R1["auth — Register/Login/Me"]
            R2["ai — Recommendations/Chat/Insights"]
            R3["trips — CRUD + Itinerary Generation"]
            R4["destinations — Countries/Regions/Destinations"]
            R5["admin — Dashboard/CRUD/Analytics"]
            R6["affiliates — Booking Tracking/Revenue"]
            R7["content — Blogs/Packages"]
            R8["subscribe — Newsletter"]
        end
        subgraph Services["5 Service Modules"]
            S1["gemini_service — LLM Integration"]
            S2["ai_destination_service — Destination Generation"]
            S3["image_service — 5-Source Image Pipeline"]
            S4["affiliate_service — URL Builder/Tracker/Revenue"]
            S5["generation_service — OSM POI Discovery"]
        end
        subgraph Data["Data Layer"]
            Models["10 SQLAlchemy Models"]
            Schemas["20+ Pydantic Schemas"]
            Validator["ItineraryValidator"]
            DB["SQLite / PostgreSQL"]
        end
    end

    subgraph External["External APIs"]
        Gemini["Google Gemini API"]
        Wiki["Wikipedia/Wikidata"]
        Pexels["Pexels API"]
        OSM["OpenStreetMap Nominatim"]
        Overpass["Overpass API"]
    end

    Frontend -->|REST JSON| Backend
    S1 --> Gemini
    S3 --> Wiki
    S3 --> Pexels
    S5 --> OSM
    S5 --> Overpass
```

4. Database Schema (10 Models)

DIAGRAM (MERMAID NOTATION)

```
erDiagram
    Country ||--o{ State : has
    State ||--o{ Destination : has
    Destination ||--o{ Attraction : has
    Destination ||--o{ DestinationTip : has
    User ||--o{ Trip : creates
    User ||--o{ AnalyticsEvent : generates
    User ||--o{ BookingClick : clicks

    Country {
        int id PK
        string name UK
        string code
        string currency
        string image
    }
    State {
        int id PK
        string name
        string image
        int country_id FK
    }
    Destination {
        int id PK
        string name
        string slug
        string desc
        text description
        string image
        string location
        string price_str
        int estimated_cost_per_day
        float rating
        string tag
        json highlights
        json itinerary
        json activities
        json gallery_images
        json best_time_months
        json vibe_tags
        int state_id FK
    }
    Attraction {
        int id PK
        string name
        text description
        int entry_cost
        string duration
        float rating
        string type
        int destination_id FK
    }
    DestinationTip {
        int id PK
        text tip
        string category
        bool is_ai_generated
        int destination_id FK
    }
    DestinationRequest {
        int id PK
        string name
        text description
        int cost
        string tag
        string status
        datetime created_at
    }
    Trip {
        string id PK
        string title
        string destination_country
        string start_city
        int budget
        int duration
        int travelers
        string style
        string date_type
        json itinerary_json
        int total_cost
        int user_id FK
    }
    User {
        int id PK
        string email UK
        string password_hash
        string name
        datetime created_at
    }
    AnalyticsEvent {
        int id PK
        string event_type
        json event_data
        int user_id FK
        datetime created_at
    }
    BookingClick {
        int id PK
        string link_type
        string destination
        string partner
        string tracking_id UK
        float estimated_revenue
        bool is_converted
        int user_id FK
    }
    Subscriber {
        int id PK
        string email UK
        bool is_active
        datetime subscribed_at
    }
```

5. Core Features — Deep Breakdown

5.1 🤖 AI-Powered Itinerary Generation (The Heart of the App)

Flow: User selects destinations → Sends preferences → Gemini generates day-by-day plan → Validator auto-corrects → Image enrichment → Response.

Pipeline Steps:

- 1. **Prompt Construction** (`gemini_service.py`):
- Loads style-specific prompt templates (`base_template.txt` , `budget_trip.txt` , `luxury_trip.txt` , `multi_city.txt`)
- Injects user preferences: origin, country, budget, duration, style, dates, interests, traveler type
- Formats selected destination data for context injection
- Applies traveler-type behavioral hints (solo male, solo female, couple, family, etc.)
- 2. **Gemini API Call** (`_generate_content_http`):
- Uses HTTP REST calls (not SDK) for `responseSchema` support
- Enforces **structured JSON output** via Pydantic schema flattening
- Model fallback chain: `gemini-2.0-flash` → `gemini-2.0-flash-lite`
- Temperature control per function
- Retry logic with exponential backoff
- 3. **Validation Layer** (`validation.py` — `ItineraryValidator`):
- **Budget Check**: Verifies total_cost within ±5% of user budget; auto-scales all costs proportionally if over
- **Activity Count Check**: Flags days with >5 activities as overpacked
- **Generic Name Detection**: Regex patterns catch lazy AI output like "local market" or "beach" without specifics
- **Cost Consistency**: Verifies individual day totals sum to ~total_cost (15% tolerance)
- **Required Fields**: Ensures `trip_title` , `total_cost` , `itinerary` exist
- 4. **Image Enrichment** (`image_service.py`):
- Attaches real images to itinerary days and the trip header
- Uses `image_keyword` from AI response as search term
- Fallback to Unsplash default if no keyword
- 5. **Analytics Logging**: Every generation (success/failure/validation) logged to `AnalyticsEvent` table

Pydantic Schema (`schemas.py`) enforces this structure for Gemini output:

Schema	Key Fields
`TripPlan`	trip_title, total_cost, cost_breakdown, itinerary[], travel_between_cities[], smart_insights, packing_tips
`ItineraryDay`	day, date, location, theme, pacing_level, activities[], accommodation, day_total, travel_hours, intensity_score
`ActivityList`	time, time_range, activity, description, why_this_fits, local_secret, cost, how_to_reach, crowd_level, meal_type, google_maps_search_query
`AccommodationConfig`	name, type (enum), cost_per_night, location, why_this, booking_tip
`InterCityTravel`	day, from_city, to_city, method, duration, travel_class, cost, alternative

5.2 🏗️ AI Destination Architect

What it does: Users can suggest new destinations. The system uses AI to validate and enrich destination details before admin approval.

- `DestinationRequest` model tracks submissions (pending → approved → rejected)
- AI generates full destination profiles via `ai_destination_service.py`
- SQLite-based **caching layer** (60-day TTL) prevents redundant API calls
- Model fallback chain with 3 retries per model
- Response validation: normalizes costs (`_parse_cost` handles "₹500", "Free", "500-1000"), checks required fields
- Auto-attaches images via the image pipeline

5.3 🌍 Smart Destination Discovery (OSM Integration)

`generation_service.py` provides an alternative, data-driven destination discovery:

- 1. **Geocode** city name → (lat, lon) via Nominatim
- 2. **Fetch POIs** within 5km radius via Overpass API (tourism attractions, historic sites, parks, museums)
- 3. **Score** each POI based on tags (Wikipedia link = +5, Wikidata = +3, website = +2, etc.)
- 4. **Parallel image fetch** for top 10 scored POIs using ThreadPoolExecutor
- 5. **Store** as `Destination` records in the database

5.4 📷 5-Source Image Pipeline

`image_service.py` (660 lines) — the most sophisticated service:

Priority	Source	How
1	**SQLite Cache**	Instant lookup, 60-day TTL
2	**Wikipedia REST**	Page summary API → main image
3	**Wikidata P18**	Direct property lookup → Commons CDN
4	**Pexels**	Rate-limited search (200/hr, 20k/month) with smart query building
5	**SVG Placeholder**	Deterministic emoji-based SVG data URIs by category

Features:

- **Smart query builder**: Uses landmark knowledge map → type keywords → location context
- **Rate limiter**: Pexels calls tracked with thread-safe counter
- **Batch helper**: `get_images_batch()` with configurable delay between requests
- **Wikipedia search fallback**: If direct lookup fails, searches for article first

5.5 🎟️ Affiliate & Booking System

Complete monetization pipeline:

- `AffiliateURLBuilder` : Constructs partner-specific URLs with UTM params and tracking IDs
- `ClickTracker` : Records every click to `BookingClick` table with user agent, IP, referrer
- `RevenueCalculator` : Probabilistic revenue estimation per click

- Commission rates: Flight 3%, Hotel 5%, Activity 10%
- Conversion rate: 8%
- Average booking values: Flight ₹8,000, Hotel ₹6,000, Activity ₹2,000
- **Partners configured**: MakeMyTrip, Booking.com
- **Admin stats endpoint**: Aggregated clicks, revenue, breakdowns

5.6 🔐 Authentication System

- **JWT-based** (Flask-JWT-Extended, 7-day token expiry)
- Endpoints: `/auth/register`, `/auth/login`, `/auth/me`
- Password hashing via `werkzeug.security`
- Frontend: `AuthContext.jsx` with safe localStorage wrapper (handles cross-origin errors)
- Protected routes: Trip save, user dashboard
- Optional JWT extraction for analytics (guest users can still generate itineraries)

5.7 🛡️ Admin Control Plane

Full admin dashboard with access-key authentication:

Endpoint	Function
`POST /api/admin/verify-key`	Verify admin access key
`GET /api/admin/stats`	Dashboard overview stats
`GET/PUT/DELETE /api/admin/destinations`	CRUD destinations
`GET /api/admin/blogs`	Manage blog content
`GET /api/admin/packages`	Manage travel packages
`GET /api/admin/users`	User management
`GET /api/admin/trips`	View all trips
`GET /api/admin/requests`	Destination request queue
`POST /api/admin/requests/:id/approve`	Approve → creates Destination
`POST /api/admin/requests/:id/reject`	Reject request
`GET /api/admin/visitors`	Real-time visitor tracking
`GET /api/admin/analytics/events`	Event log viewer
`GET /api/admin/analytics/summary`	Event counts by type

5.8 💬 AI Chat Agent

- Users can chat with a travel-knowledgeable AI assistant

- Context-aware responses using destination database
- Endpoint: `/chat` → `gemini_service.chat_with_data()`
- Frontend: `ChatWidget` component (always accessible)

5.9 📅 Flexible Date Planning

Three scheduling modes:

- 1. **Fixed Dates**: Calendar integration with start/end dates
- 2. **Duration-Based**: "10 days in October" (month + duration)
- 3. **Anytime**: Exploratory planning, no date constraints

5.10 💰 Dynamic Budget Intelligence

- Real-time budget estimation from selected destinations
- Style multipliers: Budget ×0.7, Standard ×1.0, Luxury ×1.5
- Cost breakdown: Transport 20%, Accommodation 35%, Food 25%, Activities 15%, Misc 5%
- Endpoint: `/calculate-budget`
- Post-generation validation auto-scales overbudget itineraries

---

6. Frontend Architecture

Page Structure (20+ Routes)

```
/                     → Home (Hero + Features + Destinations + Blogs)
/destinations         → DestinationsPage (Browse all)
/destinations/:id     → DestinationDetails
/destination/:id      → DestinationDetailsPage (alternate)
/plan-trip            → TripPlannerPage (THE main feature)
/trip/:tripId         → TripViewerPage (saved itineraries)
/booking              → BookingPage (affiliate links)
/blogs                → BlogsPage
/blogs/:id            → BlogDetails
/packages             → PackagesPage
/packages/:id         → PackageDetails
/login                → LoginPage
/register             → RegisterPage
/dashboard            → DashboardPage (user's saved trips)
/admin/login          → AdminLogin
/admin                → AdminDashboard
/about                → AboutUsPage
/careers              → CareersPage
/press                → PressPage
/partners             → PartnersPage
/help                 → HelpCenterPage
/safety               → SafetyPage
/cancellation         → CancellationPolicyPage
/privacy              → PrivacyPolicyPage
/terms                → TermsOfServicePage
/cookies              → CookiePolicyPage
```

Component Library (16 Groups)

Component	Purpose
`Hero/`	Landing page hero section (6 files)
`Navbar/`	Global navigation bar
`Footer/`	Global footer with links
`TripPlanner/`	**17 files** — The main trip planning wizard
`Destinations/`	Destination browsing grid
`DestinationCard/`	Individual destination card
`AddDestinationModal/`	"Architect" — suggest new destinations
`Packages/`	Travel package cards
`Blogs/`	Blog card/listing components
`Booking/`	Booking flow UI
`ChatWidget/`	AI chat floating widget
`Features/`	Feature showcase section
`AboutUs/`	Company about section
`Skeleton/`	Loading skeleton animations
`LoadingOverlay/`	Full-screen loading state
`ErrorBoundary/`	React error boundary

TripPlanner Sub-Components (The Core Feature)

File	Role
`ModernItineraryView.jsx`	Main itinerary display (20KB — largest component)
`DateSelectionModal.jsx`	Calendar + flexible date picker
`BudgetSelectionModal.jsx`	Budget range + style selection
`AIDestinationDetailsModal.jsx`	AI-generated destination deep-dive
`DestinationCard.jsx`	Destination selection cards
`TripOptions.jsx`	Trip configuration panel
`ItineraryTimeline.jsx`	Day-by-day timeline view
`TripMap.jsx`	Leaflet map with route visualization
`BookingSection.jsx`	Integrated booking links
`SortableActivityItem.jsx`	Drag-and-drop activity reordering
`SortableDayColumn.jsx`	Drag-and-drop day reordering
`ActivitySwapModal.jsx`	Swap activities between days

7. API Endpoints (Complete Map)

Auth (`/auth`)

Method	Path	Auth	Description
POST	`/auth/register`	—	Create account
POST	`/auth/login`	—	Login → JWT token
GET	`/auth/me`	JWT	Current user profile

AI (`/`)

Method	Path	Auth	Description
POST	`/recommend-destinations`	—	AI recommends destinations for country/regions
POST	`/recommend-regions`	—	AI recommends regions for a country
POST	`/destination-details-ai`	—	AI deep-dive on a destination
POST	`/calculate-budget`	—	Dynamic budget calculation
POST	`/smart-insight`	—	AI trip composition insights
POST	`/chat`	—	Chat with AI agent
POST	`/generate-destination`	—	Generate new AI destination

Trips (`/`)

Method	Path	Auth	Description
POST	`/generate-itinerary`	Optional	Generate day-by-day AI itinerary
POST	`/api/save-trip`	JWT	Save trip to user account
GET	`/get-trip/:tripId`	—	Retrieve saved trip
GET	`/api/user/trips`	JWT	List user's trips
POST	`/api/feedback`	Optional	Submit itinerary feedback

Destinations (`/`)

Method	Path	Auth	Description
GET	`/countries`	—	List all countries
GET	`/regions`	—	List regions for a country
GET/POST	`/destinations`	—	List/filter destinations
GET	`/destinations/:id`	—	Destination detail + attractions
POST	`/destinations/:id/reviews`	—	Add review
POST	`/api/populate-region/:id`	—	AI-populate a region with destinations
POST	`/api/destination-request`	—	Submit new destination request

Admin (`/api/admin`)

Method	Path	Auth	Description
POST	`/api/admin/verify-key`	Admin Key	Authenticate admin
GET	`/api/admin/stats`	Admin	Dashboard stats
GET/PUT/DELETE	`/api/admin/destinations`	Admin	Full CRUD
GET	`/api/admin/users`	Admin	User list
GET	`/api/admin/trips`	Admin	All trips
GET/POST	`/api/admin/requests`	Admin	Destination requests
GET	`/api/admin/visitors`	Admin	Visitor analytics
GET	`/api/admin/analytics/*`	Admin	Event logs & summaries

Affiliates (`/api`)

Method	Path	Auth	Description
GET	`/api/book/:type`	—	Track click → redirect to partner
GET	`/api/admin/affiliate-stats`	Admin	Revenue analytics

Content (`/`)

Method	Path	Auth	Description
GET	`/blogs`	—	Blog listings
GET	`/packages`	—	Package listings

8. Data Flow — Trip Planning Workflow

DIAGRAM (MERMAID NOTATION)

```
sequenceDiagram
    participant U as User (Browser)
    participant FE as React Frontend
    participant BE as Flask Backend
    participant AI as Gemini API
    participant IMG as Image Pipeline
    participant DB as SQLite/PostgreSQL

    U->>FE: 1. Select Country
    FE->>BE: GET /countries
    BE->>DB: Query Countries
    DB-->>BE: Country list
    BE-->>FE: JSON countries

    U->>FE: 2. Choose Regions
    FE->>BE: POST /recommend-regions
    BE->>AI: Generate region recommendations
    AI-->>BE: Region list + reasoning
    BE->>IMG: Attach images
    IMG-->>BE: Enriched regions
    BE-->>FE: Regions with images

    U->>FE: 3. Select Destinations
    FE->>BE: POST /recommend-destinations
    BE->>AI: Generate destination suggestions
    AI-->>BE: Destinations + details
    BE-->>FE: Destination cards

    U->>FE: 4. Set Preferences (budget, dates, style, interests)

    U->>FE: 5. Generate Itinerary
    FE->>BE: POST /generate-itinerary
    BE->>DB: Fetch selected destination data
    DB-->>BE: Destination records
    BE->>AI: Structured prompt + schema
    AI-->>BE: Full itinerary JSON
    BE->>BE: ItineraryValidator (budget/quality checks)
    BE->>IMG: Enrich with images
    IMG-->>BE: Images attached
    BE->>DB: Log AnalyticsEvent
    BE-->>FE: Complete itinerary

    U->>FE: 6. View & Customize
    Note over FE: Drag-drop activities, swap days, view map

    U->>FE: 7. Save Trip
    FE->>BE: POST /api/save-trip (JWT)
    BE->>DB: Insert Trip record
    DB-->>BE: Trip ID
    BE-->>FE: Confirmation + share link
```

9. Environment Variables

```
# Backend (.env in /backend/)
GEMINI_API_KEY=<Google Gemini API key>
JWT_SECRET_KEY=<secret for JWT signing>
DATABASE_URL=sqlite:///travel.db          # or postgresql://...
ADMIN_ACCESS_KEY=altair-admin-2026
PEXELS_API_KEY=<optional, for image fetching>
VALIDATION_STRICT=true                    # auto-scale overbudget itineraries
SECRET_KEY=<Flask session secret>

# Frontend (.env in root)
VITE_API_URL=                             # empty for dev (uses Vite proxy), set for product
```

10. How to Run

```
# 1. Backend
cd backend
pip install -r requirements.txt
# Create .env with API keys
python app.py                             # → http://localhost:5000

# 2. Frontend
cd ..   # project root
npm install
npm run dev                               # → http://localhost:5173 (proxied to backend)

# 3. Production Build
npm run build                             # → /dist/ folder
# Backend serves /dist/ as SPA fallback
```

11. Project File Map

```
AltairGo-Intelligence/
├── backend/
│   ├── app.py                   # Flask app factory, 8 blueprints, JWT, CORS, SPA fallba
│   ├── database.py              # SQLAlchemy engine, session, init_db()
│   ├── models.py                # 10 SQLAlchemy models (Country→Subscriber)
│   ├── validation.py            # ItineraryValidator (budget, quality, consistency)
│   ├── migrate_v1_init.py       # Initial data seeding script
│   ├── requirements.txt         # flask, sqlalchemy, google-genai, flask-jwt-extended...
│   ├── routes/
│   │   ├── auth.py              # Register, Login, Me (JWT)
│   │   ├── ai.py                # 7 AI endpoints (recommend, chat, insight, generate)
│   │   ├── trips.py             # Itinerary generation, CRUD, feedback
│   │   ├── destinations.py      # Countries, regions, destinations, reviews, requests
│   │   ├── admin.py             # 15+ admin endpoints + analytics
│   │   ├── affiliates.py        # Booking redirect + revenue stats
│   │   ├── content.py           # Blog/package content API
│   │   └── subscribe.py         # Newsletter subscription
│   ├── services/
│   │   ├── gemini_service.py       # Core Gemini integration (478 lines)
│   │   ├── ai_destination_service.py # AI destination generation + cache (669 lines)
│   │   ├── image_service.py        # 5-source image pipeline (660 lines)
│   │   ├── affiliate_service.py    # URL builder, click tracker, revenue calculator
│   │   ├── generation_service.py   # OSM/Overpass POI discovery + scoring
│   │   └── schemas.py             # 20+ Pydantic schemas for AI output
│   ├── prompts/
│   │   ├── base_template.txt    # Master prompt (170 lines, few-shot examples)
│   │   ├── budget_trip.txt      # Budget-specific prompt overlay
│   │   ├── luxury_trip.txt      # Luxury-specific prompt overlay
│   │   └── multi_city.txt       # Multi-city routing prompt
│   ├── data/                    # Static data files
│   └── tests/                   # pytest test suite
│
├── src/
│   ├── App.jsx                  # React Router (20+ routes, conditional layout)
│   ├── main.jsx                 # React entry point
│   ├── config.js                # API_BASE_URL configuration
│   ├── index.css                # Global styles
│   ├── context/
│   │   └── AuthContext.jsx      # JWT auth state management
│   ├── services/
│   │   └── TripAI.js           # 9-method API client for backend
│   ├── components/
│   │   ├── TripPlanner/         # 17 files — core trip planning wizard
│   │   ├── Hero/                # Landing hero (6 files)
│   │   ├── Navbar/, Footer/     # Global layout
│   │   ├── ChatWidget/          # AI chat floating widget
│   │   ├── Destinations/, DestinationCard/
│   │   ├── Packages/, Blogs/
│   │   ├── Booking/
│   │   ├── AddDestinationModal/    # Destination Architect
│   │   ├── Skeleton/, LoadingOverlay/, ErrorBoundary/
│   │   └── Features/, AboutUs/
│   ├── pages/
│   │   ├── Home.jsx
│   │   ├── trips/                 # TripPlannerPage, TripViewerPage, DashboardPage
│   │   ├── destinations/          # Browse, detail pages (5 files)
│   │   ├── auth/                  # Login, Register
│   │   ├── admin/                 # Admin login + dashboard
│   │   ├── booking/               # Booking page
│   │   ├── blogs/, packages/      # Content pages
│   │   ├── company/               # About, Careers, Press, Partners
│   │   ├── support/               # Help, Safety, Cancellation
│   │   └── legal/                 # Privacy, Terms, Cookies
│   └── data/                      # Static data (blogs.js, destinations.js, packages.js)
│
├── public/                        # Static assets
├── vite.config.js                 # Smart proxy config (HTML bypass for SPA routes)
├── vercel.json                    # SPA rewrite rules
├── package.json                   # React 19, Vite 7, Framer Motion, Leaflet, dnd-kit
└── README.md
```

12. Key Differentiators / Unique Technical Aspects

- 1. **Structured AI Output**: Uses Gemini's `responseSchema` with flattened Pydantic schemas — not just free-text generation
- 2. **Post-Generation Validation**: Custom `ItineraryValidator` catches and auto-corrects budget overruns, generic names, and inconsistent costs
- 3. **5-Source Image Pipeline**: Cache → Wikipedia → Wikidata → Pexels → SVG — ensures every destination always has an image
- 4. **Smart Proxy Architecture**: Vite config distinguishes HTML navigation (→ React SPA) from API calls (→ Flask) on conflicting routes like `/destinations`
- 5. **Prompt Engineering**: 170-line master prompt with few-shot examples, strict rules, and budget math instructions
- 6. **OSM Integration**: Real POI discovery using Overpass API with relevance scoring algorithm
- 7. **Affiliate Revenue Layer**: Complete click tracking → estimated revenue pipeline with per-partner URL construction
- 8. **Traveler-Type Awareness**: Different prompt hints for solo male, solo female, couple, family, group, elderly travelers