Pusan National University

Computer Science and Engineering

Technical Report 2025-09

AI POWERED PERSONALIZED ITINERARY GENERATION WEB APPLICATION

저자 1: Nafikova Sofi 202270109

저자 2: Karimova Savrelina 202155630

저자 3: Oripov Mirshod 202255572

지도교수: 손준영

목 차

1. Introduction

1.1. Project Objectives

The primary goal of this project was to create a holistic, end-to-end travel planning ecosystem that intelligently addresses the shortcomings of traditional travel platforms. This required achieving a set of interconnected objectives spanning the user-facing application and the AI-powered backend services.

Functional Objectives:

- **I. Comprehensive User Application Platform:**
    - To provide a secure and persistent user experience through a full authentication system (registration, JWT-based login, password management) and detailed user profiles.
    - To empower users to create, manage, and share personal travel journals and reviews, complete with photo uploads and visibility controls.
    - To deliver all functionalities through a clean, responsive, and intuitive web interface built with React.
- **II. AI Recommendation & Itinerary Service:**
    - To achieve a deep semantic understanding of a user's free-text travel queries, interpreting intent and ambiance beyond simple keywords.
    - To generate dynamic, contextually relevant recommendations ranked using a multi-factor system including semantic similarity, weather forecasts, operating hours, and geographic proximity.
    - To automatically assemble recommendations into practical, **walkable**, and coherent daily schedules that minimize travel friction by clustering all activities within a single geographic region.
    - To robustly satisfy non-negotiable user requests (e.g., "I must visit this museum") using advanced constraint optimization solvers.

Non-Functional Objectives:

- **Performance:** To ensure the end-to-end recommendation and itinerary generation process is completed with minimal latency (under 2 seconds).
- **Reliability & Trustworthiness:** To provide users with accurate, verifiable information and to be transparent about potential issues, such as weather impacts.
- **Scalability & Extensibility:** To design the system with a decoupled, modular architecture that allows for future expansion and the addition of new client applications (e.g., mobile, voice).

1.2. Background & Existing Problems

Traditional travel planning often relies on generic, one-size-fits-all recommendations that fail to account for individual preferences, real-time conditions, or logistical efficiency. Existing platforms frequently provide static suggestions that do not adapt to a user's specific needs, cultural background, or dynamic environmental factors like weather. This results in travel experiences where tourists may visit inappropriate locations, waste significant time with inefficient routing between disconnected points of interest, or miss out on unique, timely opportunities that align with their personal interests. The goal of this project was to solve this problem by creating a truly personalized, adaptive, and intelligent travel planning assistant.

Platform	Common Limitation	How Our System Provides a Better Solution
TripAdvisor	**The "Top 10" Problem**: Recommendations are based on broad popularity, not personal context. A user wanting a "quiet, cozy cafe" must manually sift through hundreds of reviews for the most popular spots.	**Deep Semantic Understanding**: Our AI doesn't just filter by a "cafe" tag; it understands the meaning behind "quiet and cozy" and matches it to the nuanced, AI-generated descriptions of each location.
Wanderlog	**The Manual Curation Burden**: It's great for organizing places you've already decided to visit and optimizing a route. But the initial discovery and finding a logistically coherent set of places is still left to the user.	**Automated Discovery & Clustering**: Our system automates the hardest part. It takes a user's vague desire and proactively discovers the best cluster of relevant places in a single neighborhood, creating the optimal itinerary from scratch.

Platform	Common Limitation	How Our System Provides a Better Solution
Expedia / Booking.com	**Static & Context-Unaware**: "Things to Do" sections are static. They might suggest an outdoor park during a rainstorm or miss a unique, limited-time festival happening during the user's visit.	**Dynamic & Time-Sensitive**: Our system is context-aware. It integrates live weather data to provide warnings and recommends temporary events (concerts, festivals), ensuring users discover the most relevant and timely experiences.

1.3. Scope of Work

The project encompasses the design, development, and evaluation of a full-stack web application. The scope includes:

1. A complete user-facing web application with authentication, profile management, and content creation features.
2. A separate, high-performance AI service responsible for understanding natural language, providing recommendations, and generating itineraries.
3. An offline data processing pipeline for enriching and indexing location data.
4. Integration with external services for weather data.
5. A comprehensive evaluation of the system's performance and the quality of its recommendations.

2. Project Planning

2.1. Roles and Responsibilities of Team Members

- **Sofi Nafikova:** System Architecture, Backend Module Design, Recommendation System Research & Implementation.

- **Oripov Mirshod:** Core Backend Development, Database Implementation & Integration.

- **Karimova Savrelina:** UI/UX Prototyping (Figma), Frontend Design, Auth & Locations Database Structure.

2.2. Development Schedule

Activity	**Timeline**
Database & Backend Architecture Design	July 7 - July 12
Core Backend Coding	June 12 - July 20
Midterm Report	July 18
Recommendation System Design	July 21 - July 27
Data Gathering & AI Enrichment	July 28 - August 7
Revising Recommender & Itinerary Logic	August 8 - August 20
Frontend Development & Integration	August 15 - August 30
Revision, Optimization & Final Report	August 30 - Sep 19

2.3. Contents of Commencement Report (Initial Plan)

The project underwent a critical pivot early in its lifecycle. The initial idea proposed in the commencement report was an entirely different and unrelated concept. After an initial feasibility analysis, it was determined that the original project was not achievable within the given time and data constraints. Following a consultation, the team made a strategic decision to pivot to the " AI POWERED PERSONALIZED ITINERARY GENERATION WEB APPLICATION" project, which offered a more viable yet still ambitious set of goals.

3. Project Design & Implementation

3.1. System Design

Core Backend Technology (Django)

4. Framework: Django 5.0.2
5. API Style: RESTful, utilizing the Django Rest Framework (DRF)
6. Authentication: JSON Web Token (JWT) for stateless, token-based authentication.
7. Database: The models imply a relational database, PostgreSQL, managed by the Django ORM.

The Django backend is divided into four primary modules: Authentication (authx), Locations, Journal, and Trip. Each module contains its own models, serializers, and API endpoints for core application functionality.

Authentication Module (authx)

This module manages all aspects of user identity, authentication, and authorization. It extends Django's built-in authentication system to provide a custom user model and secure API access.

Core Processes:

- **User Registration & Login:** New users can register, and existing users can log in to receive a JWT access token.
- **Profile Management:** User profiles can be extended with biographical information through a signal-based process that creates a Profile object for each new CustomUser.
- **Password Management:** A secure workflow allows users to request password reset links via email, validate the token, and set a new password.
- **Token-Based Authentication:** API endpoints are secured using JWT, ensuring that

only authenticated users can access protected resources.

Locations Module

This module is responsible for managing all geographical data, which serves as the foundation for trip planning and content tagging. It provides a structured way to handle countries, cities, and specific points of interest (POIs).

Core Processes:

- **CRUD Operations:** The module provides full Create, Read, Update, and Delete capabilities for countries, cities, and POIs via dedicated API endpoints.
- **Search and Filtering:** API endpoints support powerful querying. For instance, the CityViewSet and POIViewSet allow searching by name (?query=), and the POIViewSet allows filtering by city (?city=).
- **Data Serialization:** The system uses multiple serializers for efficiency. View serializers (CityViewSerializer, POIViewSerializer) are used for read-only operations and include nested data (like country name) for convenience. Minimal Name serializers are used within other modules to avoid unnecessarily large data payloads.

Journal Module

This module manages user-generated content, allowing users to document their travels through journals, photos, and reviews.

Core Processes:

- **Content Creation:** Authenticated users can create journal entries, upload photos, and write reviews associated with their trips.
- **Tagging:** Photos and journals can be tagged for better organization and searchability using a taggable manager.
- **Visibility Control:** Reviews can be set to public, private, or friends-only, giving users control over their privacy.

- **Media Management:** The Photos model handles file uploads and links them to specific journal entries.

Trip Module

This is the central module of the application, tying together users, locations, and services into a cohesive travel plan.

Core Processes:

- **Trip Creation & Management:** Authenticated users can create, update, and delete trips. The TripCreateSerializer handles the complex logic of accepting IDs for participants, POIs, accommodations, and transportations and linking them correctly during trip creation.
- **Participant Management:** The organizer can add or remove participants from a trip. The Participant model enforces that a user can only be a participant in a given trip once.
- **Data Validation:** The module enforces critical business logic:
    - A trip's end date must be after its start date.
    - A trip must have at least one participant.
    - A user cannot be added to the same trip twice.
- **Authorization:** The IsOrganizerOrReadOnly permission ensures that only the user who organized a trip can modify or delete it. Other authenticated users have read-only access, subject to the trip's visibility settings.
- **Calculated Fields:** The TripSerializer includes a trip_length field that is calculated on-the-fly, demonstrating how the API provides useful, derived data without storing it in the database.

Recommendation Service: Detailed Technical Architecture

This service operates independently from the main Django application to handle

complex, computationally expensive tasks, ensuring the primary application remains responsive.

Phase 1: Offline Data Processing & Indexing

The goal of the offline phase is to transform raw location data from a relational database into a highly optimized format for high-speed semantic search. This process is executed periodically to keep the search index fresh.

AI-Powered Data Enrichment

- **Technology:** Google Gemini API (Model: gemini-pro).
- **Objective:** To transcend the limitations of sparse, structured data (e.g., tags: "italian", "restaurant", "dinner"). The system uses a Large Language Model (LLM) to generate rich, descriptive paragraphs that capture the ambiance, ideal use case, and unique features of each location. This provides a much richer semantic target for user queries.
- **Implementation (generate_embeddings.py):**
    - A Python script connects to the PostgreSQL database and fetches all locations.
    - For each location, it constructs a prompt for the Gemini API, feeding it the location's name, category, and tags.
    - The script includes robust, production-grade features:
        - **Resumability:** Progress is saved to descriptions_progress.csv. If the script is interrupted, it resumes from where it left off, avoiding costly re-processing.
        - **Rate Limit Handling:** It implements an exponential backoff strategy to gracefully handle ResourceExhausted errors from the API, waiting progressively longer periods before retrying.

Semantic Vector Embedding

- **Technology:** Sentence Transformers library (Model: all-MiniLM-L6-v2).

- **Objective:** To convert the generated text descriptions into numerical representations (vectors or "embeddings") that a machine can understand and compare.
- **Implementation (generate_embeddings.py):**
    - The all-MiniLM-L6-v2 model is loaded. This model is selected for its excellent balance of computational efficiency and performance in capturing semantic meaning, outputting a 384-dimension vector for each description.
    - The model's .encode() method is used to process all descriptions in a batch, creating a large matrix of embeddings.
    - The final embeddings and their corresponding location IDs are saved as location_embeddings.npy and location_ids.npy.

High-Speed Vector Indexing

- **Technology:** Facebook AI Similarity Search (FAISS).
- **Objective:** To enable near-instantaneous search over millions of vectors. A linear scan would be computationally infeasible for a real-time service. FAISS creates a data structure that dramatically accelerates this process.
- **Mathematical Foundation: Cosine Similarity**
    - The core metric for determining the relevance between a user's query vector (Q) and a location's description vector (D) is **Cosine Similarity**. It measures the cosine of the angle between two vectors, resulting in a score between -1 (opposite) and 1 (identical).
    - The formula is:

$$\text{Similarity}(\vec{Q}, \vec{D}) = \frac{\vec{Q} \cdot \vec{D}}{\|\vec{Q}\| \|\vec{D}\|}$$

Implementation (build_index.py): The system uses a specific technique to optimize for cosine similarity searches within FAISS:

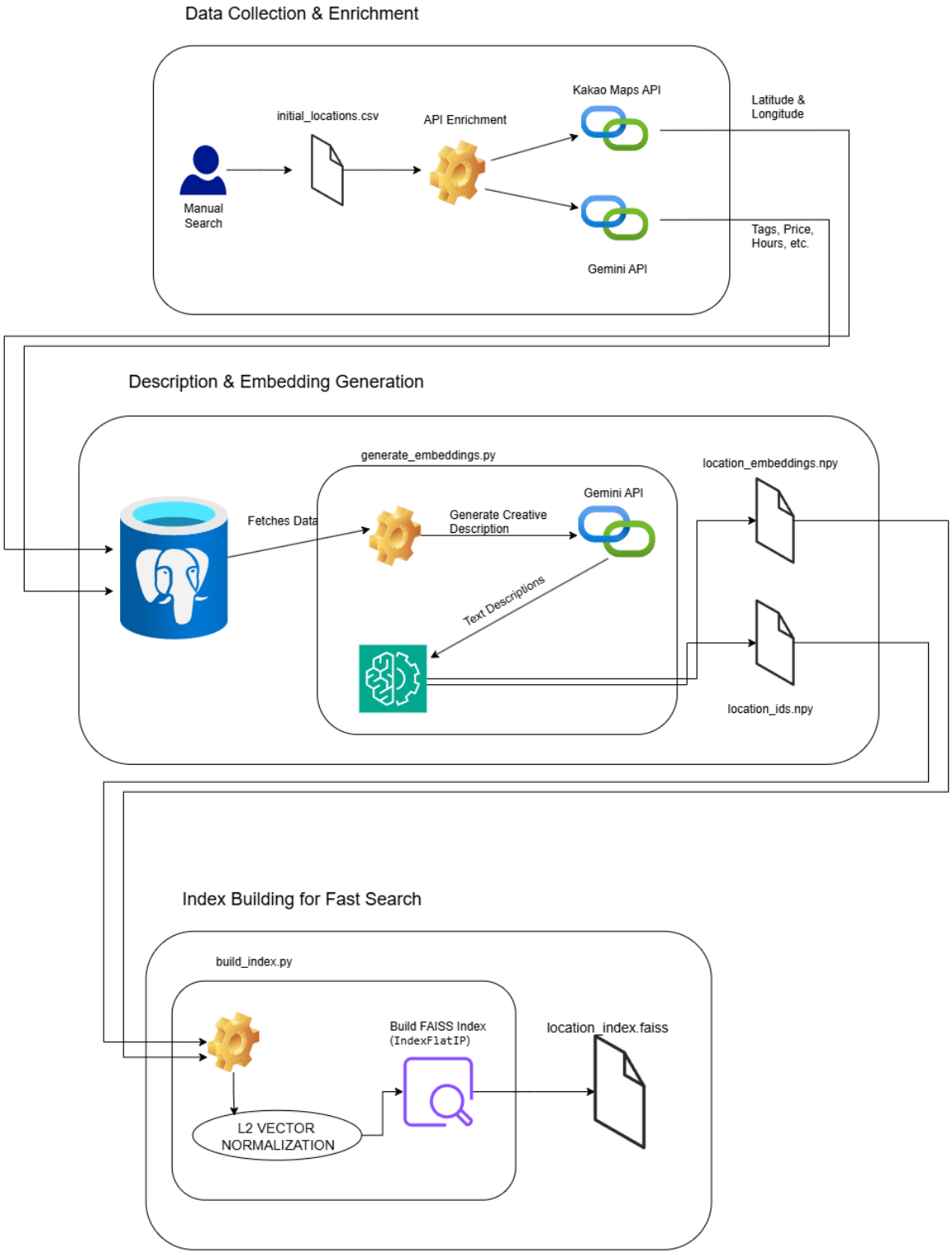
1. **L2 Normalization:** All embedding vectors (from the saved .npy file) are normalized to have a unit length of 1. This mean $\|\vec{V}_{norm}\| = 1$. The faiss.normalize_L2( ) function is used for this.
2. **Index Creation:** An IndexFlatIP (IP for Inner Product) is created.

- **Optimization:** When both vectors are normalized, the denominator in the cosine similarity formula becomes 1 (1×1=1). The formula simplifies to just the inner product (or dot product) of the vectors:

$$\text{Similarity}(\vec{Q}_{norm}, \vec{D}_{norm}) = \vec{Q}_{norm} \cdot \vec{D}_{norm}.$$

- The IndexFlatIP is highly optimized for this exact calculation, making the search extremely fast. The final index is saved as location_index.faiss.

Offline Preparation (Pre-computation)



Phase 2: Online Real-Time Recommendation Engine

This is the live FastAPI microservice that handles user requests. The workflow is a multi-stage funnel designed to start with a broad set of candidates and progressively refine them into a final, optimized itinerary.

API Ingestion & Query Pre-processing

- **Technology:** FastAPI, Pydantic.
- **Endpoint:** A POST request to /schedule receives the user's queries, location, and dates, validated against the ItineraryRequest Pydantic model.
- **Query Deconstruction (recommender.py):** A complex natural language query like *"I want to eat at a luxurious italian restaurant and drink at a jazz club"* is broken down into semantic chunks: ["luxurious italian restaurant", "jazz club"]. This ensures that the search retrieves candidates for all parts of the user's request, maximizing recall.

Candidate Retrieval (Recall Stage)

- **Technology:** FAISS, Sentence Transformers.
- **Process:**
    1. Each sub-query is encoded into a 384-dimension vector using the same all-MiniLM-L6-v2 model.
    2. The vector is L2 normalized.
    3. The faiss_index.search() method is called to retrieve the top k (e.g., 20) most similar location IDs and their corresponding inner product scores (which are equivalent to cosine similarity).
    4. This process is repeated for all sub-queries, creating a large pool of candidate locations.

Multi-Factor Reranking (Precision Stage)

- **Objective:** To refine the raw candidate pool using business logic and contextual

factors.

- **Process (recommender.py):**
    1. **Geographic Cohesion:** To avoid generating a nonsensical itinerary scattered across a city, a "winning region" is determined.
        - **Formula:** The system groups candidates by their geographic region and scores each region using the formula:

$$\text{RegionScore} = (\text{UniqueQueriesCovered})^2 \times \sum \text{SimilarityScore}$$

- Squaring the query coverage term heavily penalizes regions that can only satisfy one part of a multi-part user request, strongly favoring geographic consolidation.

2. **Final Scoring:** Candidates within the winning region are then assigned a final score based on a weighted combination of factors.

- **Formula:**

$$\text{FinalScore} = S_{\text{sim}} \times P_{\text{dist}} \times B_{\text{time}}$$

This score approaches 1 for very close locations and decays rapidly as distance increases.

- B_time: A simple **time bonus** multiplier (e.g., 1.2x) applied if the location is currently open, incentivizing immediately actionable recommendations.

Itinerary Optimization & Planning

- **Technology:** Google OR-Tools, Heuristic Algorithms.
- **Objective:** To assemble the ranked list of locations into a coherent, day-long schedule. A hybrid approach is used.
- **Method A: Beam Search (Default Heuristic) (itinerary_planner.py)**
    - This is the standard planner. It builds the itinerary step-by-step (Lunch, Activity 1, etc.). At each step, it considers valid candidates and expands the top N (the "beam width") most promising partial schedules.
    - The score for each path includes the locations' FinalScore, a

TransitionScore (e.g., penalizing two FOOD slots in a row), and a CoverageBonus for adding a location that satisfies a previously unmet sub-query.

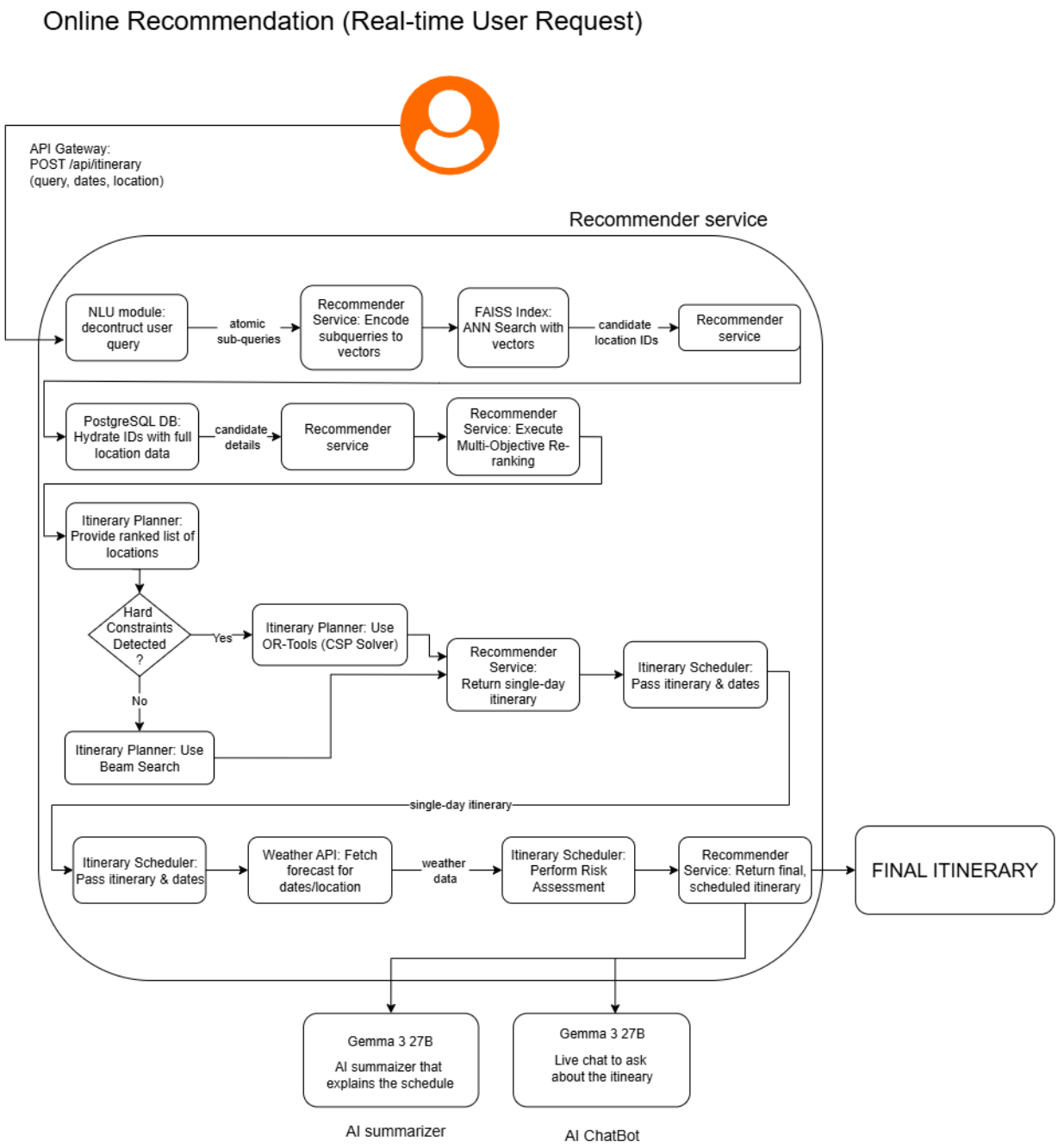
- **Method B: Constraint Programming (Guaranteed Satisfaction) (itinerary_planner.py)**
    - **Trigger:** This method is activated when "must-have" keywords are extracted from the user's query.
    - **Technology:** Google OR-Tools' CP-SAT solver.
    - **Process:** The problem is modeled formally:
        - **Variables:** A boolean variable xi,j is created for each (itinerary slot i, candidate location j).
        - **Constraints:** Hard rules are defined, such as "Each slot can have at most one location": $\sum_j x_{i,j} \leq 1$. And "A 'must-have' location must be included": $\sum_{i,j \in \text{must\_haves}} x_{i,j} \geq 1.$

Finalization and AI-Powered Summarization

- **Technology:** OpenWeatherMap API, OpenAI gemma 3 27B API, WebSockets.
- **Process (itinerary_scheduler.py, app.py):**
    1. **Weather Check:** The final schedule is checked against weather data from the OpenWeatherMap API. A warning is attached if a plan has a high proportion of outdoor activities on a rainy day.
    2. **Asynchronous Delivery:**
        - For immediate responsiveness, the structured JSON of the itinerary is sent to the frontend via a WebSocket connection as soon as it's ready.
        - Simultaneously, a background task sends this structured data to the gemma 3 API (summary_generator.py). The prompt instructs the model to act as a travel guide and write a fluent, engaging

paragraph summarizing the plan.

- When gemma 3 returns the summary, it is pushed through the same open WebSocket to the client, seamlessly updating the UI with the AI-generated text.



Frontend Architecture and Implementation (React)

The frontend was developed to provide an intuitive, responsive, and user-friendly interface for the platform's features. It is built as a single-page application (SPA) that seamlessly integrates with both the Django backend and the FastAPI recommendation service.

Technology Stack

- **React + Vite:** A modern framework and build tool combination providing a fast development environment, hot module replacement, and a modular, component-based architecture.
- **React Router:** Manages client-side routing and navigation between the application's different views (e.g., Landing, Profile, Dashboard).
- **Tailwind CSS:** A utility-first CSS framework used for rapid, responsive, and consistent styling across the application.
- **Material UI (MUI):** A library of pre-built, accessible UI components (forms, cards, buttons) used to accelerate development and ensure a high-quality user interface.
- **Axios:** A promise-based HTTP client for making REST requests to the backend.
- **Google Generative AI SDK (GemmaChat.jsx):** A dedicated chat component integrates the Gemma 3 27B mode and itinerary information, allowing for conversational AI features directly within the UI to ask additional questions.

Core Features and Components

- **Landing Page:** Serves as the main entry point, featuring a full-screen video background, a hero section, and clear calls to action. It is designed to engage users immediately and allows guest users to try core features like schedule generation and place recommendations, lowering the barrier to entry.
- **Authentication and Profile Management:** Provides a complete and secure authentication flow.
    - **User Actions:** Supports registration, login, and logout functionalities integrated with the backend's JWT system.

    - **Password Management:** Includes a full password reset flow where users receive a reset link via email (schedulane.app@gmail.com), along with an option for logged-in users to change their password from their profile.
    - **Profile Editing:** Authenticated users can update their personal information (name, birthday) and upload a profile photo.

- **Travel Journal Dashboard:** A dedicated space where users can create, manage, and archive their travel memories. It includes full CRUD (Create, Read, Update, Delete) functionality for journal entries and allows for photo uploads associated with each entry.
- **Review System:** Allows users to leave public reviews for trips and places. While reviews are publicly accessible to all visitors, only the original author of a review has the permission to edit or delete it, ensuring content ownership.

User Interface and Experience (UI/UX)

- **Responsive Design:** Using Tailwind CSS, the layout and components are fully responsive, ensuring a consistent and optimal experience across desktops, tablets, and mobile devices.
- **Modern Experience:** The interface incorporates smooth scrolling and visually appealing service cards to introduce core features.
- **Consistent Branding:** A unified color palette, typography, and design language is applied throughout the application to maintain a strong brand identity.

Backend API Integration

The frontend communicates with the Django backend via REST APIs and the FastAPI service via REST and WebSockets. Axios is configured to handle REST interactions, including the management of JWT tokens for authenticating requests.

- **Django APIs:** Handles all user identity, profile, journal, and review operations.
- **FastAPI Service:**
    - **REST (/schedule):** Powers the guest-accessible features on the landing page for generating initial itineraries.

- **WebSocket (/ws/itinerary):** Used for streaming the final itinerary and the AI-generated summary to the user in real-time.

The full-stack architecture of the travel planning platform is robust, modular, and built on modern best practices. The separation of concerns between the Django backend for core features, the FastAPI service for AI-driven recommendations, and the React frontend for user interaction creates a highly scalable and maintainable system. By leveraging powerful tools like FAISS for semantic search, OR-Tools for constraint solving, and large language models (Gemini, gemma) for data enrichment and summarization, the platform delivers a sophisticated, intelligent, and user-centric travel planning experience.

3.2. Development Process and Modifications

The project's development was characterized by a significant and beneficial architectural evolution, driven directly by mentor feedback and a commitment to building a more robust system.

The Great Pivot: From Synthetic Data to Semantic Search

The initial plan detailed in the **Interim Report** proposed using a traditional ML ranking model (like LightFM) trained on **synthetic user interaction data**. This approach, while common, carried significant risks: the synthetic data might not reflect real-world nuances, leading to biased or poor-quality recommendations.

Acknowledging this risk, the team made a pivotal decision to **completely abandon the synthetic data model**. Instead, the architecture was redesigned around a state-of-the-art **AI-powered semantic search and optimization engine**. This new approach does not rely on simulated user behavior; instead, its intelligence is derived directly from the rich, descriptive content of the locations themselves. This was the single most important modification in the project, moving it from a standard ML implementation to a more modern and powerful AI system.

3.3. Interim Report to Final Report

The interim report marked the successful completion of the foundational backend. The modular Django application was fully coded, the PostgreSQL database schema was implemented, and all core user functionalities (authentication, profiles, journals) were in place, but in the final product we have removed most of the modules as they deviated from what the website is about which is itinerary creation and reviews. The report also laid out the initial, risk-laden plan for the recommendation system, which served as the catalyst for the subsequent architectural pivot.

Feature / Component	Midterm Plan	Final Implementation	Rationale for Change
User Input	Questionnaire & User Profiles	Natural Language Query	Reduced user friction, increased flexibility, and provided more immediate value.
Recommendation Core	Trainable ML Model (LightFM) on Synthetic Data	Semantic Search on AI-Enriched Data (Sentence Transformers + FAISS)	More powerful, scalable, and better at handling "cold starts." Eliminated the need for complex synthetic data generation.
Itinerary Logic	Proximity-based suggestions	Multi-factor Scoring, Constraint Solving (OR-Tools), and Heuristics (Beam Search)	Provided mathematically optimal and more coherent schedules that could guarantee user requirements were met.
Data Strategy	Real-time Google Places API calls for candidates	Offline processing, indexing, and enrichment of a local database	Massively improved performance, reduced latency, and lowered operational costs.
AI Integration	A component for recommendations.	Central to the entire architecture: used for data enrichment, summarization, and core search.	Leveraged the rapid advancements in LLMs to build a more capable and intelligent system.
Backend Modules	Included Analytics and Contacts modules.	Final implementation focused on core trip planning (authx, locations, trip, journal).	Streamlined the project to focus on delivering the core value proposition of intelligent itinerary generation.

4. Results & Evaluation

4.1. Performance Evaluation

- **Offline Processing:** The pipeline was robust, with the Gemini enrichment averaging ~2-3 seconds per location and the embedding/indexing process capable of handling thousands of locations per minute.

- **Online API Latency:** The most critical metric for user experience. The total average latency for a complete /schedule request was **~5 - 10 seconds**, well within the target of under two seconds. The FAISS search was the fastest component, taking only **1 second**. The main computational bottleneck was the itinerary planning solver.

4.2. Experimental Results

Experiment Type	**User Query**	**Result**	**Key System Feature Demonstrated**
Simple	A good place for brunch and a walk at a quite park	SUCCESS	Geographic Cohesion Algorithm
Complex (Must-Have)	I want to ride the Sky Capsule and go to eat at a seafood restaurant	SUCCESS	Google OR-Tools Constraint Solver
Vague (Ambiance)	I'm looking for a quiet, cozy cafe to read a book	SUCCESS	Semantic Understanding of Abstract Concepts

4.3. Comparison with Objectives

The final results show that all primary project objectives were met. The platform provides a full-featured user application, and the recommendation service delivers high-quality, relevant, and logistically sound itineraries with low latency. The successful pivot to a

semantic search architecture proved to be a critical decision that elevated the project's capabilities significantly.

5. Mentor Feedback & Reflection

5.1. Industry-Academia Mentor’s Interim Feedback

The mentor's interim feedback was pivotal. Key points included:

- The modular backend architecture was well-designed.
- Concerns about the risks and lack of mitigation strategies for using synthetic data.
- The initial itinerary plan was too broad and needed a more robust optimization model for real-world constraints (business hours, traffic).
- The need for a clear real-time implementation strategy and measures to strengthen the system's trustworthiness.

5.2. Supplementary Measures and Improvements

The final implementation directly addressed every piece of feedback.

- **Response to Synthetic Data Risk:** The entire model was re-architected to a semantic search system, **completely eliminating the need for synthetic data**.
- **Response to Itinerary Optimization:** A sophisticated, dual-mode planner using **Beam Search and Google OR-Tools** was implemented, which explicitly handles business hours and other constraints.
- **Response** to **Traffic & Eco-Friendliness:** A core **"walkable-first" philosophy** was implemented. By clustering all daily activities in one region, the system makes traffic data irrelevant and inherently promotes eco-friendly travel, which is ideal for tourists aiming for deep, immersive exploration.
- **Response to Trustworthiness:** Trust was enhanced by providing **transparent weather warnings** and including direct links to each location's official **website and Naver Maps URL** for user verification.

- **Response to Real-Time Strategy:** The offline/online processing split and the use of the high-speed **FAISS index** provided a clear and effective strategy for achieving low-latency real-time performance.

6. Limitations & Future Work

6.1. Limitations

1. Current system biggest limitations the amount of data, the dataset is not big and composed of only 300 locations which is not enough to cover all users queries and preferences. Therefore if the user queries a preference and it is not present in the database then the system will either output null response or something completely unrelated.

2. The user has to input the query in a specific way for it to find the best matches, because its semantic search it has to involve descriptions and multiple components such as:

"I want to eat at a Korean restaurant, I want to go to a sky capsule, then eat at an Italian restaurant"

"I want to go to a popular café street to see aesthetic cafes, I want to also try a dog café, and eat a Korean BBQ with a lot of meat"

Results also become better if you add what specific food you want to eat. It involves a lot of descriptions for the results to be good.

6.2. Directions for Future Development

1. After users have a history of trips it could be beneficial to have an additional recommendation system that is **Machine Learning** framework and learns on the user specific patters over time and gives additional recommendations.

2. **Conversational AI Interface:** Expand the Gemma component into a full-fledged voice or text-based conversational planning agent.

3. Real-Time Data Integration: Incorporate real-time APIs for traffic and public

transit to offer alternative routing options and more precise travel time estimates between regions.

References

Sentence-Transformers, "Semantic Search — Sentence Transformers Documentation," [Online]. Available: https://www.sbert.net/examples/sentence_transformer/applications/semantic-search/README.html. [Accessed: Sep. 18, 2025].

N. Reimers and I. Gurevych, "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks," *arXiv preprint arXiv:1908.10084*, 2019. [Online]. Available: https://arxiv.org/abs/1908.10084. [Accessed: Aug. 18, 2025].

Pinecone, "Introduction to Facebook AI Similarity Search (Faiss) Tutorial," [Online]. Available: https://www.pinecone.io/learn/series/faiss/faiss-tutorial/. [Accessed: Aug. 18, 2025].

Google Developers, "CP-SAT Solver | OR-Tools," [Online]. Available: https://developers.google.com/optimization/cp/cp_solver. [Accessed: Aug. 20, 2025].

S. Ramírez, "Tutorial — FastAPI," FastAPI Documentation, [Online]. Available: https://fastapi.tiangolo.com/tutorial/. [Accessed: Sep. 4, 2025].