Final Project - Midterm Report

Pusan National University

Computer Science and Engineering

2025-1

ML-Powered Personalized Tour Guide Web Application

Team: BYTE

Nafikova Sofi 202270109

Oripov Mirshod 202255572

Karimova Savrelina 202155630

지도교수: 손준영

We completely changed our project due to data and time constraints. We consulted with professor and decided to change our project with more achievable goals.

I. Project goals – Change

Traditional travel planning often relies on generic recommendations that fail to account for individual preferences and environmental factors. Most existing travel platforms provide static suggestions that do not adapt to users' specific needs, age demographics, cultural backgrounds, or circumstances such as weather. This results in suboptimal travel experiences where tourists may visit inappropriate locations, waste time with inefficient routing, or miss opportunities that align with their personal interests.

The goal is to develop a comprehensive ML powered tour guide web application that can analyze user preferences, generate customized itineraries using machine learning algorithms, and make better recommendations based on weather conditions.

A lot of time goes into researching and planning trips, which can be very time consuming, therefore it's needed to have an automated system that makes trip planning easier. The plan is that it will recommend 2 main activities a day and multiple secondary activities close by to the main activities.

Specifics:

a. **Implement a questionnaire for user input.** Question the user's personal details such as their country of origin, age, city they are visiting, how many days the trip will be, preferences such as which food do they prefer, what type of restaurants, budget, etc. This is to start

b. **Utilize Machine Learning Recommendation System**. Create a custom-trained recommendation system using synthetic data that can suggest relevant activities, restaurants, and locations based on user preferences. The system will rank suggestions by relevance, proximity, and preference alignment.

c. **Enable Geographic Clustering**. Design algorithms that cluster recommended locations geographically.

d. Integrate Weather-Aware Scheduling. Itinerary adaptation based on weather forecasts, planning between indoor and outdoor activities based on conditions.

e. **Achieve Context-Aware Planning**. Implement a system that highlights 2 main locations per day while suggesting complementary nearby activities, creating a balanced and well-structured itinerary

f. **Develop Interactive Web Interface**. Create a responsive web application featuring an interactive map with the locations pinned, display the trip's plan with details, user account with saved trips and favorite places, etc.

g. Modular Backend Architecture
Django-based backend with clearly separated modules:

- authx for user authentication (JWT, password reset, profile)
- locations for managing countries, cities, POIs

- services for restaurants, hotels, and other businesses
- reviews and bookings modules for future extensions
- planner or itinerary module to compute and save travel plans
- NLP + ML integration for parsing user input and generating itinerary logic

II. Target Problem and Requirements Analysis - Change

The problem this project aims to solve is the lack of truly personalized, adaptive, and independent travel planning systems that can provide real-time, intelligent recommendations.

Requirements Analysis

User Input Processing System	Questionnaire, basic question-answer form. If feasible in the future, might apply NLP for extraction of key entities. - destination city, trip duration (1-7 days), activity preferences (outdoor/indoor, cultural, adventure, food, nightlife), age and country of origin, dietary restrictions and accessibility needs
Machine Learning Recommendation System	Custom-trained recommendation system using synthetic training data - The system must rank locations by relevance to user preferences, popularity and ratings, weather appropriateness, time and budget constraints - Support for diverse activity types: restaurants, museums, parks, beaches, shopping, nightlife, cultural sites, adventure activities
Weather-Aware Scheduling	Integration with weather API for real-time forecast data - Automatic adaptation of recommendations based on weather conditions: sunny weather → outdoor activities prioritized, rainy weather → indoor alternatives suggested
Daily itinerary	Daily itinerary structure with 2 main locations per day and supporting nearby activities. Transportation recommendations and walking distance from maps APIs

Sophisticated PostgreSQL for Data Storage and User Management with its backend	PostgreSQL database for users' info and their trips: - Users: authentication system with: User profile, JWT authentication with customized tokens, Password management - Profile - user behavior, time spent, session tracking - Reviews - write about a tri and attach photos., Users can rate a trip (1–10), write pros/cons, and control visibility (public, private, friends). - Trips - duration, visited countries, hotel/lodging info, Includes fields like name, location, amenities, check-in/out, and price - Expenses - manage travel-related expenses for each trip: category: type of expense, amount and currency, trip: each expense is linked to a trip, details: optional text, created: timestamp - User preferences and interests
Interactive Web Interface	Responsive design compatible with web on desktop and mobile. Interactive map display showing all recommended locations. Detailed location information including descriptions and highlights, estimated visit

Non-Functional Requirements

Performance	Efficient caching of weather data and location information. Real-time recommendation generation fast enough
Scalability	Modular architecture allows easy addition of new destinations. API rate limiting and resource management
Reliability	Backup recommendation systems in case of ML model failures. Enough uptime requirement with proper error handling
Security	Secure API endpoints with proper authentication. Data encryption for sensitive user information

III. Analysis of Real-World Constraints and Countermeasures – Change

a. Training Data Limitations and Quality

Constraint: Creating comprehensive synthetic training data for diverse destinations and user preferences requires significant effort and may not capture all real-world scenarios or cultural nuances.

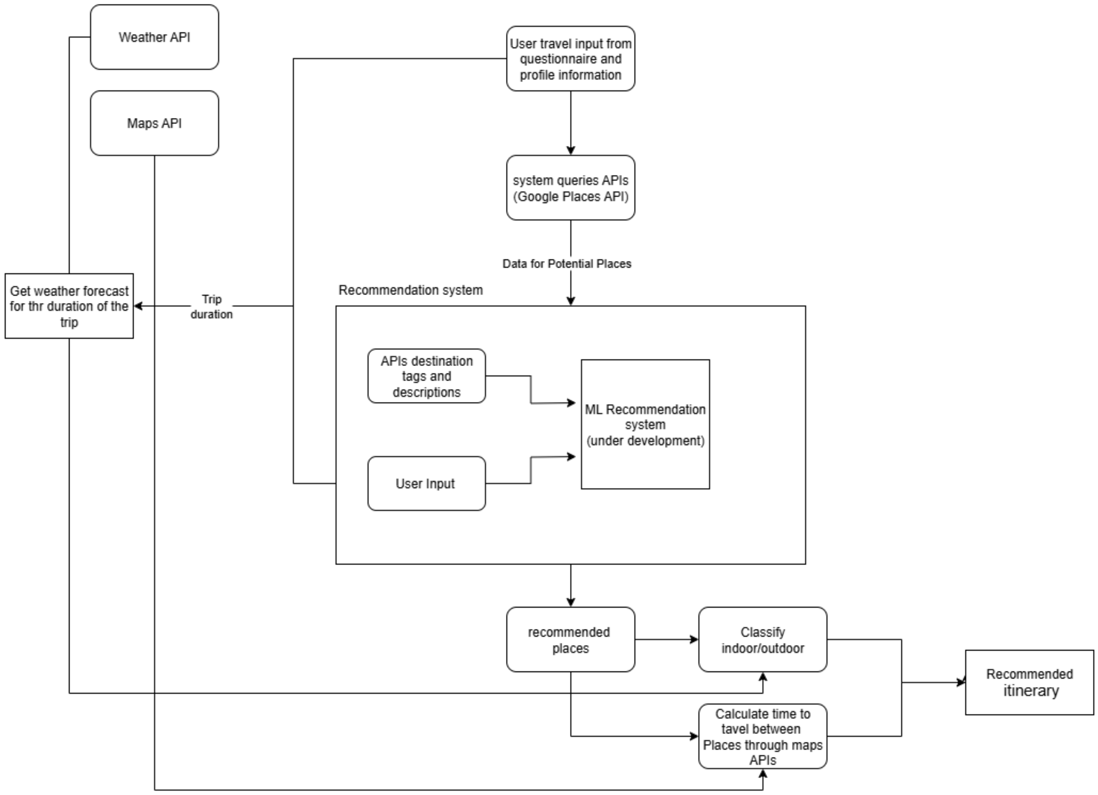
b. Weather API Dependencies and Costs

Constraint: Reliance on external weather APIs creates potential points of failure and ongoing costs that may impact system reliability and budget constraints.

c. Model Performance and Accuracy

Constraint: Custom-trained models may not achieve the same accuracy as established third-party APIs, potentially leading to poor recommendations and user dissatisfaction.

IV. Schedule generation design documentation – change


Weather API
Maps API
User travel input from questionnaire and profile information
system queries APIs (Google Places API)
Data for Potential Places
Recommendation system
Get weather forecast for thr duration of the trip
Trip duration
APIs destination tags and descriptions
ML Recommendation system (under development)
User Input
recommended places
Classify indoor/outdoor
Recommended itinerary
Calculate time to tavel between Places through maps APIs

Initial - subject to change depending on project progress and further research. More explanation at the bottom of the report

V. Timeline

Activity	Date	Progress
Designing database	June 22 – June 24	Done
Designing base backend architecture	June 22 – June 28	Done
Coding the base backend architecture	June 28 – July 13	Done
Designing frontend	June 24 – July 13	In progress
Midterm Report	July 13 – July 18	Done
Designing ML system	July 21 – July 27	
Gathering data and creating synthetic data	July 28 – August 7	
Training ML model	August 8 – August 20	
Integrating ML model into the backend system (coding)	August 21 – August 14	
Coding the frontend	August 15 – August 21	
Creating final itinerary recommendation page	August 15 – August 21	
Connecting the backend with the frontend	August 22 – August 30	
Revision and optimization	August 30 – Sep 17	
Final Report	Sep 19	

VI. Progress

Team member	Progress
Sofi Nafikova	- Developed System Architecture - Designed databases architecture and backend modules design - Researching possible ML models
Oripov Mirshod	- Designed database structure - Coded the core backend - Connected databases to the project
Karimova Savrelina	- Designed prototype UI/UX in Figma - Design database structure for auth and locations module

Backend System Architecture Progress and the backend itself

Technology stack

Layer	Tech	Purpose
Framework	Django 5.0.2	Main web framework
API Layer	Django REST Framework (DRF)	RESTful API endpoints
Auth System	CustomUser + JWT (SimpleJWT)	Authentication, profile, secure endpoints
Maps APIS	APIS	Google maps, naver maps, kakao maps API for fetching locations, places and descriptions

Database

Tool	Purpose
PostgreSQL	Production-grade RDBMS
Django ORM	Model abstraction and migrations
Migrations System	Built-in Django migrations

Authentication and Security

Component	Purpose
CustomUser model	Extended fields like profile photo, birthdate
JWT	Stateless token-based authentication
Django Permissions	Ensures object-level access
Django REST Password Reset	For secure email reset functionality

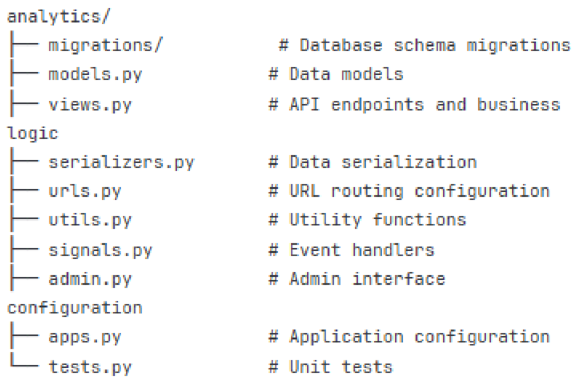
Backend project architecture is organized into multiple reusable apps (modules). Each one is responsible for a specific domain of the system:

Analytics module

The analytics module follows Django's Model-View-Template (MVT) architecture pattern and consists of the following key components:

- Data Layer - Models for tracking user activities and analytics data
- Business Logic Layer - Views containing analytical computation logic
- API Layer - REST API endpoints for data consumption
- Signal Processing - Event-driven session management
- Utility Functions - Helper functions for complex calculations

Module Structure (all modules follow this structure):



```
analytics/
├── migrations/           # Database schema migrations
├── models.py            # Data models
├── views.py             # API endpoints and business
logic
├── serializers.py       # Data serialization
├── urls.py              # URL routing configuration
├── utils.py             # Utility functions
├── signals.py           # Event handlers
├── admin.py             # Admin interface
configuration
├── apps.py              # Application configuration
└── tests.py             # Unit tests
```

Data Models

UserActivity Model

The primary data model tracks user session information:

Fields:

user: Foreign key to CustomUser (CASCADE deletion)

start_time: Automatically set session start timestamp

end_time: Session end timestamp (nullable)

Key Features:

Automatic session start time recording

Manual session termination capability

Relationship with custom user authentication system

API Endpoints Architecture

Analytics Categories

The module provides 11 distinct API endpoints organized into the following categories:

1. Review Analytics

- Endpoint: /most-liked-review/
- Functionality: Returns top 5 most liked public reviews
- Access: Public (AllowAny)

2. Financial Analytics

- Endpoint: /expense-analytics/<trip_id>/
- Functionality: Trip-specific expense analysis with category breakdown
- Access: Authenticated users only

3. Content Popularity Analytics

  - Endpoints:
    - /popular-pois/ - Most visited Points of Interest
    - /popular-accommodations/ - Most booked accommodations
  - Access: Public (AllowAny)

4. Trip Pattern Analytics

- Endpoints:
  - /average-trip-duration/ - User-specific trip duration analysis
  - /timeline/ - Current year trip timeline
  - /most-visited-countries-organizer/ - Personal country visit statistics
  - /most-visited-countries/ - Global country visit statistics

5. User Analytics

- Endpoints:
  - /total-time-spent/ - Individual user session time analysis
  - /total-users/ - Platform user count statistics

Data Privacy

- User Isolation: Personal analytics are user-specific
- Public Data Filtering: Only public reviews are included in analytics
- Session Security: Automatic session termination on logout

Integration Architecture

Cross-Module Dependencies

The analytics module integrates with multiple platform modules:

- Authentication Module: authx.models.CustomUser
- Trip Management: trip.models.Trip
- Location Services: locations.models.POI, City, Country
- Financial Management: expense.models.Expense
- Review System: journal.models.Review
- Service Management: services.models.Accommodation
-

All the modules follow the same architecture with some little modifications. Below is a brief description of the rest of the modules.

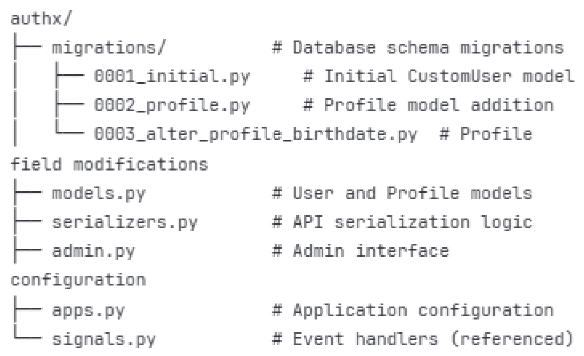
Authentication Module Architecture

The authx module is a Django-based authentication system that provides comprehensive user management capabilities for the travel planning platform. This module implements a custom user authentication system with extended profile management, JWT token-based authentication, and secure password management features.

Core Components

The authentication module is built upon Django's authentication framework and consists of the following key components:

1. Custom User Model - Extended user authentication with additional fields
2. Profile Management - User profile extension with biographical information
3. JWT Authentication - Token-based authentication system
4. Password Management - Secure password handling and validation
5. Serialization Layer - API data serialization and validation
6. Signal Processing - Event-driven profile management
7. Admin Interface - Administrative user management



```
authx/
├── migrations/          # Database schema migrations
│   ├── 0001_initial.py     # Initial CustomUser model
│   ├── 0002_profile.py     # Profile model addition
│   └── 0003_alter_profile_birthdate.py  # Profile
field modifications
├── models.py            # User and Profile models
├── serializers.py       # API serialization logic
├── admin.py             # Admin interface
configuration
├── apps.py              # Application configuration
└── signals.py           # Event handlers (referenced)
```

CustomUser Model

The core authentication model extends Django's AbstractUser:

Field Structure:

- Identity Fields:
  - username: Unique identifier (max 100 chars)
  - email: Unique email address (max 255 chars)
  - first_name: User's first name (max 150 chars)
  - last_name: User's last name (max 150 chars)
- Access Control Fields:
  - is_active: Account activation status
  - is_admin: Administrative privileges
  - is_staff: Staff access permissions

Authentication Configuration:

- USERNAME_FIELD: Set to "username"
- REQUIRED_FIELDS: ["email"]
- Inherits from AbstractUser and PermissionsMixin

Security Best Practices Implementation

Password Management

- Validation: Django's password validators
- Hashing: Secure password hashing

- Confirmation: Password confirmation requirements
- Reset Security: Secure password reset workflow

Data Validation

- Input Sanitization: Comprehensive field validation
- Type Safety: Proper field type definitions
- Constraint Enforcement: Database and application-level

Password Reset

- reset_password_request_token: Sends email with reset link.
- reset_password_validate_token: Validates token.
- PasswordResetConfirmAPIView: Confirms new password using token.

Contacts Module

Module that manages friendship connections

Field	Type	Description
id	BigAutoField	Primary Key (auto-increment)
user	ForeignKey → CustomUser	The user initiating the friendship
friend	DateTimeField	Timestamp when the friendship was created

unique_together = ("user", "friend”) Prevents duplicate friendships.

Relationship Logic

User Role	Relationship Type	Related Name in Django ORM
Initiator	user → CustomUser	user_friendships
Friend	friend → CustomUser	friend_friendships

Validations

Validation Rule	Message

Cannot add yourself as a friend	"Cannot add yourself as a friend."
Cannot add the same friend twice	"Friendship already exists."

API Functionality

HTTP Method	Route	Functionality
POST	/contacts/friendships/	Add a new friend
GET	/contacts/friendships/	List user's friends, with optional query param
DELETE	/contacts/friendships/{id}/	Remove a friendship

Serializer Behavior

- Create/Update: FriendshipSerializer
- Read/GET: FriendshipReadSerializer (includes nested friend info via CustomUserSerializer)

Expense Module

Field	Type	Description
id	BigAutoField	Primary key
trip	ForeignKey → Trip	The trip this expense is associated with
category	CharField (choices)	Type of expense (e.g., accommodation, food, etc.)
amount	DecimalField(10, 2)	Monetary amount of the expense
currency	CharField(max_length=3)	3-letter currency code (e.g., USD, EUR)
details	TextField(blank=True)	Optional text description of the expense
created	DateTimeField(auto_now_add)	Timestamp when the expense was recorded

Category choices: accommodation, transportation, food, point of interest, activity, other.

Relationship to Trip: Each Expense is linked to a specific Trip, using a foreign key with on_delete=models.CASCADE, meaning that Deleting a Trip will also delete all its associated expenses.

API functionality

HTTP Method	Endpoint	Description

POST	/expense/	Create a new expense
GET	/expense/	List all expenses
GET	/expense/{id}/	Retrieve single expense
PUT/PATCH	/expense/{id}/	Update an existing expense
DELETE	/expense/{id}/	Delete an expense

Journal Module

This module handles user-generated travel content: journals, reviews, and photos. It includes tagging, visibility controls, ratings, and media uploads.

TravelJournal

Field	Type	Description
id	BigAutoField	Primary Key
trip	ForeignKey → Trip	Linked trip
user	ForeignKey → CustomUser	Entry author
title	CharField(max_length=100)	Journal title
notes	TextField	Full journal content
tags	TaggableManager	Tags for the entry
created	DateTimeField(auto_now_add)	Creation time
updated	DateTimeField(auto_now=True)	Last updated time

Relations: Many-to-one with Trip, User

Photos

Field	Type	Description
id	BigAutoField	Primary Key
photo	FileField	Uploaded photo file
journal_entry	ForeignKey → TravelJournal	Associated journal entry
tags	TaggableManager	Tags for search/organization
created	DateTimeField(auto_now_add)	Upload timestamp

Review

Field	Type	Description
id	BigAutoField	Primary Key
user	ForeignKey → CustomUser	Reviewer
trip	ForeignKey → Trip	Trip being reviewed
rating	PositiveIntegerField (1–10)	Numeric rating

comment	TextField	Main review text
likes	PositiveIntegerField (default=0)	Like count
dislikes	PositiveIntegerField (default=0)	Dislike count
recommended	BooleanField	Was the trip recommended?
pros	TextField(blank=True)	Optional positives
cons	TextField(blank=True)	Optional negatives
visibility	CharField(max_length=10)	public / private / friends
date	DateTimeField(auto_now_add=True)	Submission timestamp
created	DateTimeField(auto_now_add=True)	Redundant with date, likely for sorting

API Endpoints via ViewSets

Model	Endpoint	Notes
TravelJournal	/journal/travel-journal/	CRUD for journal entries
Photos	/journal/photos/	Photo upload & management
Review	/journal/reviews/	Ratings with visibility options

Services Module

This module manages accommodations and transportation options that can be linked to trips. It provides CRUD APIs for each, searchable and scoped by the creator.

Field	Type	Description
id	BigAutoField	Primary Key
name	CharField(100)	Hotel name
location	CharField(100)	City/location
details	TextField(blank=True)	Description
price_per_night	DecimalField(max_digits=10, 2)	Cost per night
checkin_date	DateField()	Start date
checkout_date	DateField()	End date
amenities	TextField(blank=True, null=True)	Optional amenities
created_by	ForeignKey → CustomUser (nullable)	Creator of entry (user)
created	DateTimeField(auto_now_add=True)	Timestamp

total_price(): returns total price based on number of nights.

Transportation

Field	**Type**	**Description**
id	BigAutoField	Primary Key
name	CharField(100)	Transport name

departure_location	CharField(100)	Starting point
arrival_location	CharField(100)	Destination
departure_time	DateTimeField()	Departure datetime
arrival_time	DateTimeField()	Arrival datetime
description	TextField()	Description
price	DecimalField(max_digits=10, 2)	Price
type	CharField(choices=car/bus/train/plane/other)	Type of transportation
created_by	ForeignKey → CustomUser (nullable)	Creator of entry
created	DateTimeField(auto_now_add=True)	Timestamp

ViewSets and API Logic

Common Features:

- Custom Query Filtering via ?query=<name> for both models.
- Scoped by created_by, so users only access their entries.

Disables Pagination using a custom class: NoPagination.

AccommodationViewSet

- Lists, creates, and modifies user's own accommodations.
- Custom query filtering for name search.
- Automatically assigns created_by.

TransportationViewSet

- Same logic as accommodation.
- Includes all travel-related fields and filtering.

Locations Module

The locations module manages geographic entities used throughout the travel planner, including countries, cities, and points of interest (POIs). It provides fully-featured CRUD APIs and is optimized for search, filter, and integration into itinerary planning.

Country

Field	Type	Description
id	BigAutoField	Primary key
name	CharField(100, unique)	Country name
iso_code	CharField(2)	ISO alpha-2 code
currency	CharField(50)	National currency
primary_language	CharField(50)	Language

flag	ImageField	Uploaded flag image

Reverse Relation:cities: all City objects related to the country.

City

Field	Type	Description
id	BigAutoField	Primary key
name	CharField(100, unique)	City name
country	ForeignKey → Country	Belongs to a country
population	PositiveIntegerField	Optional population data
latitude	DecimalField(9, 6)	Optional latitude
longitude	DecimalField(9, 6)	Optional longitude
region	CharField(100)	Optional region or state

Reverse Relation: pois: all POI objects located in this city.

POI (Point of Interest)

Field	Type	Description
id	BigAutoField	Primary key
name	CharField(100)	POI name
description	TextField(blank=True)	Optional description
city	ForeignKey → City	Belongs to a city
location_latitude	DecimalField(9, 6)	Optional latitude
location_longitude	DecimalField(9, 6)	Optional longitude
opening_hours	CharField(200)	Optional opening hours

Serializers

Serializer	Model	Purpose & Notes
CountrySerializer	Country	Full details
CountryNameSerializer	Country	Only shows id and name (used in nested fields)
CitySerializer	City	For create/update
CityViewSerializer	City	Read-only + nested country name
CityNameSerializer	City	Minimal serializer with only id, name
POISerializer	POI	For create/update
POIViewSerializer	POI	Read-only with nested city name

ViewSets & API Logic

CountryViewSet

- Endpoint: /locations/countries/
- Features:
    - Searchable by ?query=
    - Sorted by name
    - Uses CountrySerializer

CityViewSet

- Endpoint: /locations/cities/
- Features:
    - Searchable by ?query=
    - Auto-switches between CitySerializer (write) and CityViewSerializer (read)
    - Sorted by name

POIViewSet

- Endpoint: /locations/pois/
- Features:
    - Search by ?query=
    - Filter by city with ?city=<name>
    - City slug is matched case-insensitively
    - Auto-switches between POISerializer (write) and POIViewSerializer (read)
    - Sorted by city then name

Progress Result

We have designed and developed the core backend and databases of our application, the core infrastructure that will later be used with ML system.

The Django backend is composed of modular, well-separated apps that work together to power a full-featured travel planner. Each module serves a specific domain, but together they form a cohesive ecosystem:

- The analytics module for tracking user activities and analytics data

- The authx module is a Django-based authentication system that provides comprehensive user management capabilities for the travel planning platform
- Journal Module handles user-generated travel content: journals, reviews, and photos. It includes tagging, visibility controls, ratings, and media uploads.
- Contacts module is for managing friends
- Services Module manages accommodations and transportation options that can be linked to trips
- The locations module manages geographic entities used throughout the travel planner, including countries, cities, and points of interest (POIs).

Future deliverables

We will create a scheduler module that will be generating custom and personalized itineraries with an ML model system that will be trained on synthetic data.

APIs for locations and places in the city

API	Description	Limits
Google Places API	This lets you search for places (e.g. "museums in Paris") and returns many results with full details.	up to 100,000 requests/day, but rate limits apply. Up to **20 results per page**, max **60 total** with pagination.
Nearby Search API	To find places around a point	
Place Details API	Fetch more info about places	

Google Places API will be used to fetch 20-60 places in the city that the person will have a vacation in to choose the main activities for the user to do.

Nearby Search API will be used to fetch secondary places around the main activities.

Google Places API example output for searching places in New York

```
{
 "places": [
  {
   "displayName": "Joe’s Pizza",
   "formattedAddress": "7 Carmine St, New York, NY 10014, USA",
   "geometry": {
    "location": { "latitude": 40.730610, "longitude": -73.935242 }
   },
   "place_id": "ChIJN1t_tDeuEmsRUsoyG83frY4",
   "types": ["restaurant", "point_of_interest", "establishment"]
  },
  ...
 ]
 ...
}
```

Up to **20 results per page**, max **60 total** with pagination.

ML Recommendation System:

The recommendation system operates through a two-stage process:

1. **Primary Recommendation Phase**: Upon user input of preferences and location details, the backend retrieves 60 places via location APIs. The ML system analyzes these options and predicts user preferences to generate initial recommendations.
2. **Secondary Recommendation Phase**: After primary destinations are selected, the Nearby Search API fetches an additional 60 secondary locations. The ML system performs a second prediction cycle to recommend complementary destinations.

Machine Learning System Specifications

Problem Definition

Multi-modal Recommendation/Ranking System

Input Parameters:

- User profile data (age, nationality, preferences, food preferences)
- Place characteristics (name, category, rating, reviews, description)
- Historical user interaction data from similar user profiles

Output:

- Ranked list of locations ordered by predicted user preference scores

We will be evaluating and testing different open-source models to see which one will be the best fit for our goal output

Open source models we will test:

1. LightFM (Hybrid Recommendation)

- **Architecture**: Matrix factorization with user/item metadata integration
- **Advantages**: Addresses cold start problems effectively through demographic and preference features; enhances recommendation quality for users with interaction history through collaborative filtering

2. Factorization Machines (FM) & Field-aware FM (FFM)

- **Architecture**: Generalized linear model for recommendation systems
- **Advantages**: Efficiently processes categorical and continuous feature interactions; performs well with sparse datasets and cold start scenarios

Possible Fallback Systems - depending on circumstances one of these will be chosen

1. Sentence Transformers
   Implementation: Cosine similarity matching between user preferences and location descriptions
   Model: sentence-transformers/all-mpnet-base-v2

2. Large Language Model
   Implementation: Gemini-2.5-pro for location recommendations
   Provides contextual recommendations based on user preferences

Weather forecasting for making activities match to the weather

- Implementation: Gemini-2.5-pro LLM for intelligent weather-location matching
- Functionality:
  - Analyzes forecasted weather conditions for the target location and timeframe
  - Categorizes recommended places as indoor or outdoor activities
  - Dynamically adjusts recommendations based on weather suitability
  - Prioritizes indoor venues during adverse weather conditions
  - Promotes outdoor activities during favorable weather

Datasets for ML – synthetic data example structure

User Data (users.csv)

```
user_id,age,nationality,preferences

u001,28,USA,"museums,seafood,outdoor"

u002,35,Korea,"shopping,indoor,cafes"

u003,22,Germany,"historical,museums,temples"

...
```

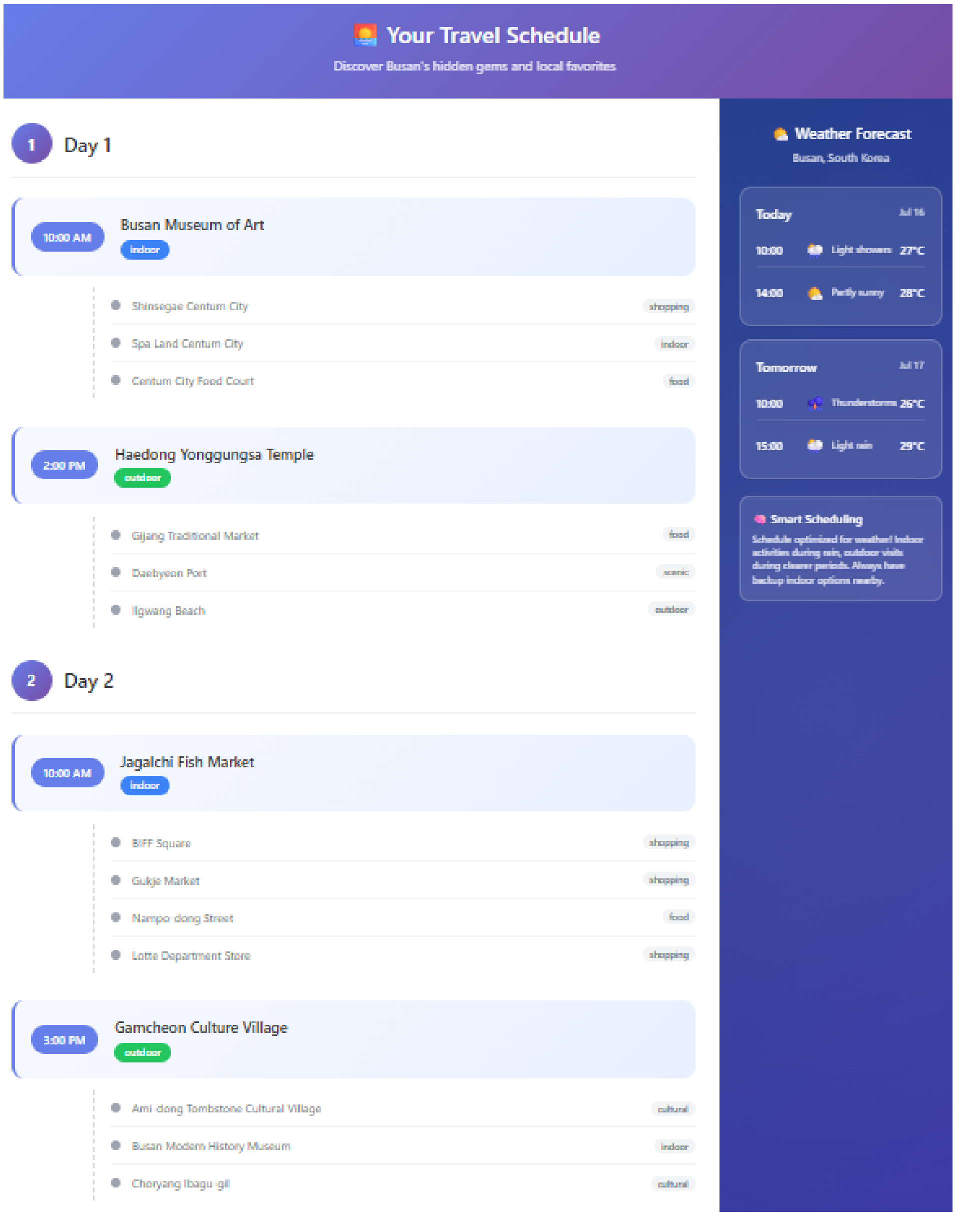
Place Data (places.csv)

```
place_id,name,type,rating,num_reviews,city,description

p001,Louvre Museum,museum,4.8,5234,Paris,"Famous art museum with Renaissance exhibits"

p002,Blue Ocean Seafood,restaurant,4.2,845,Busan,"Popular seafood spot near the beach"

p003,Nature Mall,shopping_mall,4.0,1230,Seoul,"Modern mall with restaurants and stores"

... Interactions (interactions.csv)
```

Interactions (interactions.csv)

```
user_id,place_id,liked,visited_time

u001,p001,1,2024-08-01

u002,p003,1,2024-08-02

u001,p002,0,2024-08-04

u003,p001,1,2024-08-03

...
```

Initial design how the generated itinerary will look:



🌅 Your Travel Schedule

Discover Busan's hidden gems and local favorites

1 Day 1

10:00 AM Busan Museum of Art indoor

● Shinsegae Centum City	shopping
● Spa Land Centum City	indoor
● Centum City Food Court	food

2:00 PM Haedong Yonggungsa Temple outdoor

● Gijang Traditional Market	food
● Daebyeon Port	scenic
● Ilgwang Beach	outdoor

2 Day 2

10:00 AM Jagalchi Fish Market indoor

● BIFF Square	shopping
● Gukje Market	shopping
● Nampo-dong Street	food
● Lotte Department Store	shopping

3:00 PM Gamcheon Culture Village outdoor

● Ami-dong Tombstone Cultural Village	cultural
● Busan Modern History Museum	indoor
● Choryang Ibagu-gil	cultural

⛅ Weather Forecast

Busan, South Korea

Today			Jul 16
10:00	🌦	Light showers	27°C
14:00	⛅	Partly sunny	28°C

Tomorrow			Jul 17
10:00	⛈	Thunderstorms	26°C
15:00	🌦	Light rain	29°C

🧠 Smart Scheduling

Schedule optimized for weather! Indoor activities during rain, outdoor visits during clearer periods. Always have backup indoor options nearby.