WeTravel - Design and Implementation Documentation

Group D3; April 16, 2024

Version: 1.0

Shuyang Song 1155173859, Department of Statistics
Yanze Yang 1155210986, Department of Computer Science and Engineering
Sriroth Poonyapat 1155205059, Department of Computer Science and Engineering
Ziqin Wei 1155173761, Department of Computer Science and Engineering
Thuwanontha Kittiphan 1155190410, Department of Computer Science and Engineering

Contents

1 Version History

Version	Modified by	Date	Comments
0.1	All	March 10, 2025	First Draft
0.2	All	March 29, 2025	Added License Management
0.21	All	April 3, 2025	Fixed Formatting
0.3	All	April 10, 2025	Modified contents based on development
1.0	All	April 16, 2025	Final Version

2 Summary

In this document, we have revisited and consolidated the primary requirements originally specified in the Software Requirements Specification. These requirements encompass the core user workflows, group collaboration features, and overarching system qualities necessary to ensure a reliable and secure travel planning platform. Specifically, we examined:

- **User-based requirements:** Account registration and login; profile creation and editing.
- **Group-based requirements:** Creating and joining trip groups; posting trips; expense tracking within groups.
- **Performance requirements:** Responsive filtering and feed updates (less than 2 seconds); AI recommendations within 5 seconds.
- **Security requirements:** Sensitive data (e.g., passwords) are encrypted as bcrypt hashes and stored inside database.
- **Robustness requirements:** The application must gracefully handle invalid inputs (e.g., malformed trip data, missing required fields) by displaying clear, user-friendly error messages.

3 Data Models

3.1 Overview of Data Models

This section defines the data structure and transformation from user input to final output. The system handles user profiles, group preferences, trip summaries, AI-generated recommendations, expenses, and post interactions. MongoDB is used for persistence due to its flexibility and document-based schema.

3.2 Database Schema and Collection Definitions

Collection Name	Description
Users	Stores user credentials, personal information, and preferences.
Groups	Stores group details including members, trip summary, and messages.
Trips	Stores planned trips including destination, budget, schedule, and activities.
Recommendations	Stores AI-generated trip plans based on user or group preferences.
Expenses	Stores group expenses including category, payer, and amount.
Posts	Stores user-generated trip posts with optional images and interactions.
Licenses	Stores system-generated license keys for feature access control.

3.3 Entity Relationships and Cardinality

- **User → Post (1:N)** Each `Post` references one User via `user_id`; a User can author many Posts.

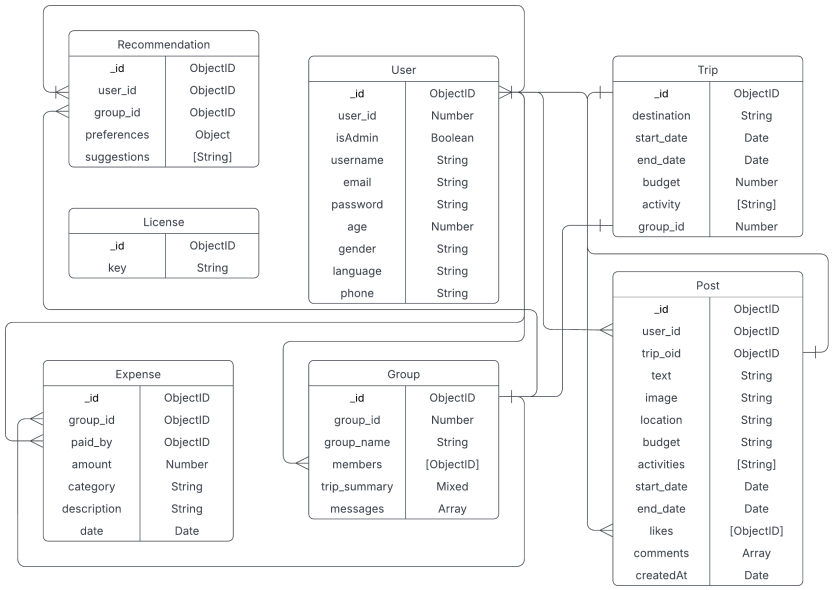

Figure 1: Entity-Relationship Diagram

- **Post → Trip (1:1)** Each `Post` stores one `trip_oid`; each Trip is created by exactly one Post.
- **User → Comment (1:N)** Comments are embedded in Posts, each with a `user_id`; a User may write many comments.
- **User ↔ Like (M:N)** `Post.likes` is an array of User _id's; Users can like multiple posts, and posts can have many likers.
- **Group ↔ User (M:N)** `Group.members` is an array of User ObjectIds; Users may belong to multiple Groups.
- **Group → Trip (1:1)** Each `Trip` has a `group_id`; by design each Group maps to one Trip.
- **Group → Expense (1:N)** `Expense.group_id` references a Group; a Group can have many Expenses.
- **User → Expense (1:N)** `Expense.paid_by` references a User; a User may pay multiple Expenses.
- **User → Recommendation (1:N)** `Recommendation.user_id` references a User; a User may request multiple Recommendations.
- **Group → Recommendation (1:N)** `Recommendation.group_id` optionally references a Group; a Group may receive many Recommendations.
- **License (no direct relationship)** `License` documents stand alone; there is currently no User↔License reference in code.

3.4 Collection Schema Definitions

User Collection

Field	Type	Description
_id	ObjectId	MongoDB document identifier.
user_id	Number	Numeric user identifier (unique).
isAdmin	Boolean	Whether the user has admin privileges.
username	String	Login and display name (unique).
email	String	Email address.
password	String	Hashed user password.
age	Number	User age.
gender	String	User gender.
language	String	Preferred language.
phone	String	Phone number.

Group Collection

Field	Type	Description
_id	ObjectId	MongoDB document identifier.
group_id	Number	Numeric group identifier (unique).
group_name	String	Name of the group.
members	[ObjectId]	Array of user _id references.
trip_summary	Mixed	Brief description of the trip.
messages	Array	Chat messages: {user_oid, text, timestamp}.

Trip Collection

Field	Type	Description
_id	ObjectId	MongoDB document identifier.
destination	String	Trip location.
start_date	Date	Trip start date.
end_date	Date	Trip end date.
budget	Number	Planned budget amount.
activity	[String]	List of planned activities.
group_id	Number	Numeric `group_id` reference.

Recommendation Collection

Field	Type	Description
_id	ObjectId	MongoDB document identifier.
user_id	ObjectId	Reference to requesting user _id.
group_id	ObjectId	Optional reference to group _id.
preferences	Object	{location, budget, duration, activities} preferences.
suggestions	[String]	Array of AI-generated suggestion strings.

Expense Collection

Field	Type	Description
_id	ObjectId	MongoDB document identifier.
group_id	ObjectId	Reference to group _id.
paid_by	ObjectId	Reference to paying user _id.
amount	Number	Expense amount.
category	String	One of {Food, Transport, Accommodation, Recreation, Other}.
description	String	Optional notes.
date	Date	Date of expense record.

Post Collection

Field	Type	Description
_id	ObjectId	MongoDB document identifier.
user_id	ObjectId	Reference to author user _id.
trip_oid	ObjectId	Reference to created Trip _id.
text	String	Post content text.
image	String	File path or URL of uploaded image.
location	String	Trip location string.
budget	String	Budget range (enum of predefined ranges).
activities	[String]	Array of activity tags.
start_date	Date	Trip start date.
end_date	Date	Trip end date.
likes	[ObjectId]	Array of user _id references who liked.
comments	Array	Array of {user_id, text, date} comment objects.
createdAt	Date	Timestamp of post creation (auto).

License Collection

Field	Type	Description
_id	ObjectId	MongoDB document identifier.
key	String	License key (unique).

3.5 Data Flow and Transformation

User Registration and Login:

1. User enters details via UI.
2. Frontend sends data to `/auth/register` or `/auth/login`.
3. Backend validates and stores or checks user credentials.
4. On success, backend returns JWT and user info.

Group Formation:

1. User selects members and inputs group details.
2. Frontend sends request to `/group/create`.
3. Backend stores group data and member IDs.

Trip Management:

1. User inputs trip details (destination, budget, date, activities).
2. Frontend sends to `/trip/create`.
3. Backend validates and links trip to the group.

Expense Logging:

1. User inputs expense data.
2. Frontend sends to `/expenses/add`.
3. Backend verifies and stores in Expense collection.

AI Recommendations:

1. User or group submits preference request.
2. Backend sends prompt to OpenAI.
3. Response is parsed and stored.

Post Creation:

1. User submits post content and optional image.
2. Image is stored via Multer and path is saved.
3. Backend stores all post fields.

License Key Management: License Key Validation:

1. Admin generates and stores a new license key.
2. User submits the license key via frontend.
3. Backend verifies the key and links it to the user if valid.
4. User gains access to restricted features.

4 Interface Design

4.1 Internal Communication

The communication between the frontend, backend, and the database.

Frontend-Backend Communication

- **Protocol:** HTTPS
- **Data Format:** JSON for input and output data

Backend-Database Communication

- **Database Structure:** MongoDB stores data in collections (Documents). The backend will use MongoDB queries to interact with the database.
- **Data Format:** JSON for input and output data.

4.2 External Communication

The communication between the web application and the external API.

Integration with ChatGPT API

- **Purpose:** To generate AI-based travel recommendations based on user experience.
- **Protocol:** HTTPS
- **Data Format:** JSON for input and output data (because we receive the input as the form).

4.3 Expected Exceptions and Handling

Frontend to Backend Communication

- **Invalid Input:** If the user submits invalid data, the backend will return an error code.
- **Session Expiry:** If the user's token expires, the backend will respond with an authentication error.

Backend to External API

- **API Downtime:** If the external APIs are unavailable, the backend will notify the user.
- **Database Connection Issue:** If MongoDB fails, the backend will notify the user.

4.4 Security Consideration

- **Data Encryption:** All sensitive user data will be encrypted using hash algorithms. Communication between the frontend, backend, and external APIs will be encrypted using HTTPS.
- **API Key Management:** API keys for external services will be stored securely.

5 Conponent Design

5.1 AuthWrapper

5.1.1 Login

- Responsibilities:
  - Mannage authentication state
  - Redirects users based on login status
  - Provides authentication context
- Inputs
  - User Credentials: email account and password
- Outputs: IsLoggedIn(Boolean Type), User Profile
- Algorithm:
  1. Check if the user is authenticated.
  2. If not, redirect to the login/signup page and instruct user to 'Forgot Password' button
  3. Fetch and store user details on successful login

5.2 Travel Group Components

5.2.1 GroupList

- Responsibilities:
  - Displays a list of available and recommended travel groups.

  - Filter and sorts groups based on user preferences.

- Inputs
  - Groups[](Array of group objects)
  - searchQuery
- Outputs: Rendered list of groups
- Algorithm:
  1. Fetch groups from API.
  2. Filter based on search query.
  3. Render ‘GroupCard’ components for each group.
  4. Sort posts based on time order from the latest to the oldest.

5.2.2 GroupCard

- Responsibilities:
  - Displays summary details of a group.
- Inputs
  - Groups[](Array of group objects)
  - searchQuery
- Outputs: Rendered list of groups
- Algorithm:
  1. Fetch groups from API.
  2. Filter based on search query.
  3. Render ‘Groupcard’ componets for each group.

5.2.3 GroupDetails

- Responsibilities:
  - Displays all details of a group.
  - Show expenses, members and chat.
- Inputs

  - GroupID

- Outputs: Rendered group details page.
- Algorithm:
  1. Fetch group details from API using GroupID.
  2. Show group info, expenses, members and chat.

5.3 Chat & Messaging Components

5.3.1 ChatBox

- Responsibilities:
  - Real-time chat for group members.
  - Supports text messages.
- Inputs
  - GroupID
  - UserID
  - Messages[]
- Outputs: OnSendMessage(message).
- Algorithm:
  1. Fetch gchar history for GroupID.
  2. Display messages using ChatMessage Components.
  3. Capture user input and send messages.

5.3.2 ChatMessage

- Responsibilities:
  - Displays individual chat messages
  - Shows sender info and timestamps.
- Inputs
  - Message sender, text, timestamp
- Outputs: Rendered chat bubble.

6 User Interface Design

6.1 Review & Rating Components

6.1.1 ReviewList

- Responsibilities: Show users reviews for trips
- Inputs: Reviews[] (String array)
- Outputs: Rendered list of reviews in text.

6.2 Page Layouts and Navigation Flow

The following is a breakdown of key pages, their components, and navigation paths.

6.2.1 Login Page

- Components
  - Username and password input fields
  - Can handle registration too
- Navigation
  - Successful login allow access to Navbar

6.2.2 Navbar

- Components
  - Navigation links
- Navigation
  - Click ”Profile” $\rightarrow$ Profile Page
  - Click ”My Trip” $\rightarrow$ Trip Management Page
  - Click ”AI Recommendations” $\rightarrow$ AI Recommendations Page
  - Click ”Login” $\rightarrow$ Login Page

6.2.3 Profile Page

- Components
  - Editable Fields: age, gender...

- Navigation
  - Via Navbar

6.2.4 AI Recommendations Page

- Components
  - Form: Preferences (destination, budget, activities, travel style)
  - ”Generate Recommendations” button
  - Results Section: Cards with generated suggestions
- Navigation
  - Via Navbar

6.2.5 Trip Management Page

- Components
  - Trip List: Displays groups and trips
  - Trip Creation Form: Filters (destination, budget, preference)
  - Invitations Section: Pending invites with ”Accept/Reject” buttons
- Navigation
  - Via Navbar

6.2.6 Admin Dashboard

- Components
  - Tabs: ”Users”
  - Users Tab: Listing users and can delete them
- Navigation
  - Auto-redirect if logged in as admin, cannot access otherwise

7 Assumptions

7.1 Technical Constraints

- Single User Login per Account: Only one user can log into the same account simultaneously. Concurrent logins will terminate existing sessions.
- Local Webpage Hosting: The application will run locally during development and testing.
- External API Dependency: The AI-driven recommendations feature requires a valid OpenAI/ChatGPT API key. The system assumes the API is accessible, responsive, and within rate limits.
- Browser Compatibility: The frontend is optimized for modern browsers (Chrome, Edge) and assumes users have JavaScript enabled.
- Media Storage Limitations: User-uploaded images/videos are stored in MongoDB with size limits to avoid performance degradation.

7.2 Development Environment Assumptions

- Developer Hardware: Developers use personal laptops.
- Standardized Tools
  - IDE: Visual Studio Code
  - Version Control: Git with GitHub for collaboration
- Local Database: MongoDB runs locally on developers' machines during testing.
- Frontend Frameworks: React, Vue, or similar frameworks are installed via npm.
- Third-Party Templates: UI templates are sourced from open-source projects to accelerate development.

7.3 User Environment Assumptions

- User Devices: Users access the platform via desktops/laptops browsers.
- Internet Connectivity: Users have stable internet access to load media and interact with the ChatGPT API.