**Design and Implementation Documentation**

**Group D3**

|  |  |  |
| --- | --- | --- |
| **Shuyang Song** | **1155173859** | **sysong@link.cuhk.edu.hk** |
| **Kittiphan Thuwanontha** | **1155190410** | **1155190410@link.cuhk.edu.hk** |
| **Ziqin Wei** | **1155173761** | **1155173761@link.cuhk.edu.hk** |
| **Yang Yanze** | **1155210986** | **1155210986@link.cuhk.edu.hk** |
| **Poonyapat Sriroth** | **1155205059** | **1155205059@link.cuhk.edu.hk** |

**Summary**

We covered a series of requirements back in the Software Requirements Specification, which mainly included user-based requirements, group-based requirements, and other non-functional requirements.

User-based requirements mainly include how users may modify their personal information and preferences. Group-based requirements mainly include how a formed group may schedule and share their traveling plans. Other non-functional requirements are mainly about performance, security and robustness issues.

User-based requirements example: create and log in accounts, setting user preferences, editing personal avatar

Group-based requirements examples: start a post to find other members, manage expenses, schedule a trip

Performance requirements: the software supports users’ actions quickly

Security requirements: data of users and the server are encrypted and safely stored

Robust requirements: the software can run in different platforms including desktop and mobile.

**System Architecture**

Our system architecture includes three parts: database, user interface and algorithms

**Database**

Database stores and manages all data user created and server-end data like list of tourist attractions. These data may be modified via algorithms that are set to work by users. It must have all CRUD functions: create, read, update and delete, hence it is accessible by users via the user interface. It should also include administrator accounts that can change all data for administration purpose. The database will be implemented using MongoDB.

**User Interface**

Since our project aims to create a travel grouping website, the user interface is essential to provide users with ways to interact with the software. It covers things like sharing posts, managing user information and managing group traveling plans. It should be accessible and appropriately decorated.

User interface serves as a way for users to create new data and modifying existing data stored in the database, and it allows users to start algorithms like recommendation of tourist attractions and posting user’s trip plans.

**Algorithms**

Algorithms handle and process all requests from users. Important algorithms employed in the software include the recommending algorithm based on external chatbot APIs, and trip posting algorithm that matches members with the same traveling destination and similar preferences. Other algorithms include functions that let users modify data in the database, functions that let users manage a group, functions that let users look up traveling information of their choices.

These algorithms may be activated via the user interface by users and they are able to modify the database.

**Data Models**

**Overview of Data Models**

This section defines the data structure, data flow, and how data is transformed from end-user input to final output. The system handles user profiles, group preferences, trip summaries, AI-generated recommendations, expenses, and post interactions. The database system is MongoDB, a NoSQL solution chosen for its scalability and flexibility.

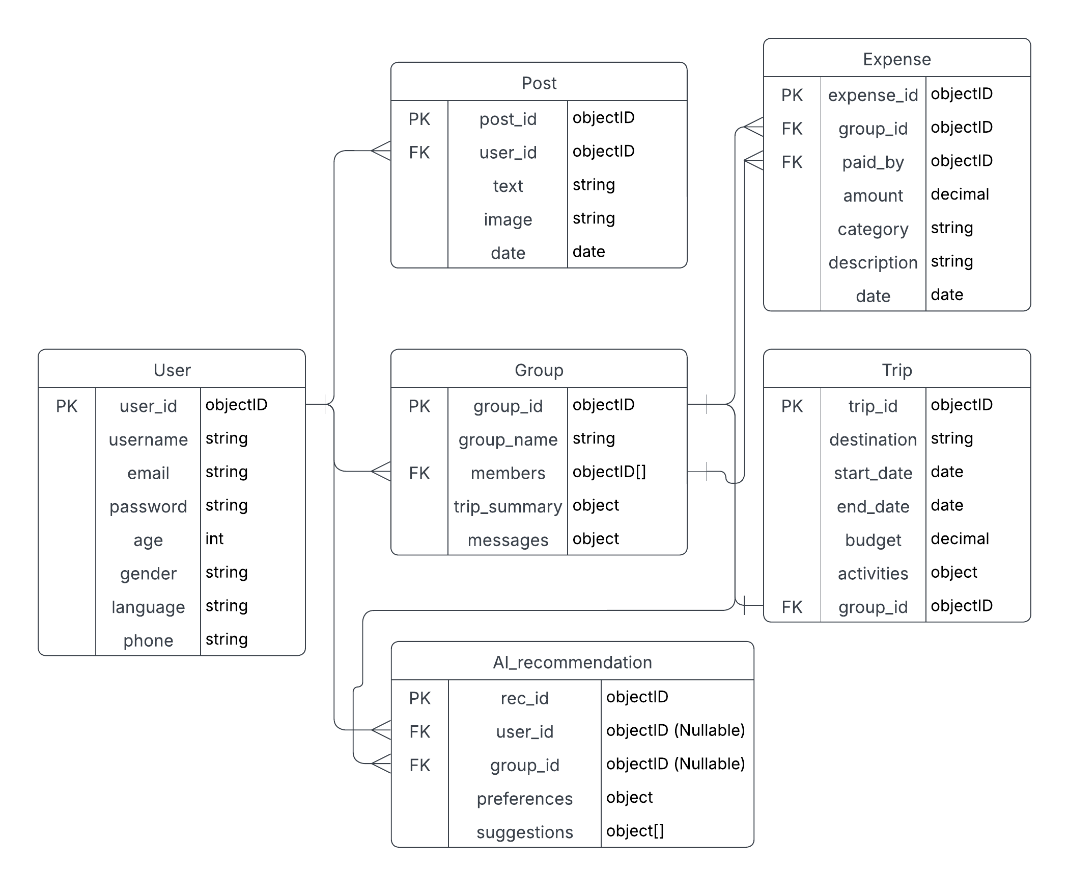
**Database Schema**

**Database Schema & Collection Definitions**

The database consists of six main collections, with relationships as follows:

|  |  |
| --- | --- |
| **Collection Name** | **Description** |
| Users | Stores user credentials, personal information, and preferences. |
| Groups | Stores group details, including members, group name, trip summary, and messages. |
| Trips | Stores planned trips, including destination, budget, start and end dates, and activities. |
| AI\_Recommendations | Stores AI-generated trip suggestions, based on either a user (individual recommendation) or a group (group-based recommendation). |
| Expenses | Stores group expenses, including category, amount, user who paid, and related group. |
| Posts | Stores user-generated content such as text, images, and post timestamps. |

**Entity-Relationship Diagram (ER Diagram)**



**Relationships and Cardinality Explanations**

1. User → Post (One-to-Many): A User can create multiple Posts.
2. User → Group (One-to-Many): A User can be part of multiple Groups.
3. User → AI\_Recommendation (One-to-Many, Optional): A User can receive individual AI-generated Recommendations before joining a Group.
4. Group → Trip (One-to-One): Each Group can have only one Trip at a time.
5. Group → Expense (One-to-Many): Each Group can have multiple Expenses.
6. User → Expense (One-to-Many): A User can pay for multiple Expenses.
7. Group → AI\_Recommendation (One-to-Many, Optional): Each Group can have many AI-generated Recommendations.

**Collection Schema Definitions**

This section defines the schema structure for each collection in MongoDB, detailing the primary attributes, data types, and descriptions.

**User Collection**

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| user\_id | ObjectID | Unique identifier for each user. |
| username | String | User's unique username used for authentication and display. |
| email | String | Email address used for authentication and notifications. |
| password | String | Encrypted password stored securely. |
| age | Integer | User's age, useful for grouping preferences. |
| gender | String | User's gender, optional for personalization. |
| language | String | User's preferred language for interactions. |
| phone | String | User's phone number for verification and notifications. |

**Group Collection**

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| group\_id | ObjectID | Unique identifier for each group. |
| group\_name | String | The name assigned to the group. |
| members | Array of ObjectIDs | Array of user\_id (of type ObjectIDs) representing group members. |
| trip\_summary | Object | Object containing summary details of the trip. |
| messages | Object | Object storing chat messages exchanged within the group. |

**Trip Collection**

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| trip\_id | ObjectID | Unique identifier for each trip. |
| destination | String | Destination of the planned trip. |
| start\_date | Date | Start date of the trip. |
| end\_date | Date | End date of the trip. |
| budget | Decimal | Estimated or planned budget for the trip. |
| activities | Object | Object containing details of planned activities. |
| group\_id | ObjectID | Foreign key referencing the associated Group.\_id. |

**AI\_Recommendation Collection**

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| rec\_id | ObjectID | Unique identifier for each AI-generated recommendation. |
| user\_id | ObjectID (Nullable) | Foreign key referencing the User.\_id (Nullable if the recommendation is for a group). |
| group\_id | ObjectID (Nullable) | Foreign key referencing the Group.\_id (Nullable if the recommendation is individual). |
| preferences | Object | User or group preferences used to generate recommendations. |
| suggestions | Array of Objects | List of AI-generated suggestions based on the preferences. |

**Expense Collection**

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| expense\_id | ObjectID | Unique identifier for each expense record. |
| group\_id | ObjectID | Foreign key referencing the Group.\_id to which the expense belongs. |
| paid\_by | ObjectID | Foreign key referencing the User.\_id who paid for the expense. |
| amount | Decimal | The total amount of the expense. |
| category | String | The category of the expense (e.g., Food, Transport, Accommodation). |
| description | String | Additional details or description of the expense. |
| date | Date | Date when the expense was recorded. |

**Post Collection**

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| post\_id | ObjectID | Unique identifier for each post. |
| user\_id | ObjectID | Foreign key referencing the User.\_id of the post creator. |
| text | String | The text content of the post. |
| image | String | URL or reference to an image attached to the post. |
| date | Date | The date when the post was created. |

**Data Flow: From End-User Input to Final Output & Data Transformation**

This section describes how data moves through the functionalities, from when a user enters input to when the processed data is stored, retrieved, and displayed through the system.

**User Registration & Login**

1. User enters registration/login details in the UI.
2. Frontend sends API request (/auth/register or /auth/login).
3. Backend validates input and checks user credentials.
   1. If registration, backend hashes password and stores user details in the User collection.
   2. If login, backend verifies credentials and generates an authentication token.
4. User details are stored in or retrieved from the User collection.
5. Backend sends authentication response (success or failure).
6. Frontend updates UI to reflect login status to user.

**Group Formation Flow**

1. User enters group details & selects members in the UI.
2. Frontend sends API request (/group/create).
3. Backend validates members. Then,
   1. New group is created in the Groups collection.
   2. Members’ user IDs and other group details are stored in the Groups collection.
4. Backend sends success response confirming group creation.
5. Frontend updates UI to display the new group to user.

**Trip Management Flow**

1. User enters trip details (destination, budget, start date, activities) in UI.
2. Frontend sends API request (/trip/create).
3. Backend validates group existence & permissions. Then,
   1. New trip with its details is created and stored in the Trip collection.
   2. Trip is linked to the group using Trip.group\_id.
4. Backend sends succuess response to confirm.
5. Frontend updates UI to display trip details to user.

**Expense Logging & Retrieval**

1. User enters expense details (amount, category, paid\_by) in UI.
2. Frontend sends API request (/expenses/add).
3. Backend validates group\_id and paid\_by (ensures payer is in a group). Then,
   1. Expense with its details is recorded in the Expense collection.
4. Backend sends success response.
5. Frontend updates UI to show updated group expenses to user.

**AI Recommendation Generation & Retrieval**

1. User or group requests AI recommendations.
2. Frontend sends API request (/ai/recommendation).
3. Backend processes request and retrieves stored preferences. Then,
   1. External AI API is called to generate recommendations.
   2. AI-generated recommendations are stored in the AI\_Recommendation collection.
4. Backend sends recommendations response.
5. Frontend updates UI to display AI-generated trip suggestions to user.

**User Posting & Interaction**

1. User writes a post & attaches an image (optional).
2. Frontend sends API request (/posts/create).
3. Backend validates post content & user authentication. Then,
   1. Post with its information is stored in the Post collection.
4. Backend sends confirmation response.
5. Frontend updates UI to display the new post in the group feed to user.

**Interface Design**

1. Internal Communication: The communicaton between the frontend, backend and the database.
   1. Frontend-Backend Communication
      1. protocol: HTTPS
      2. Data Format: JSON for input and output data
      3. REST API endpoints will be used to connect the frontend and the backend.
   2. Backend-Database Communication
      1. Database Structure: MongoDB stores data in collections (Documents). The backend will use MongoDB queries to interact with the database.
      2. Data Format: JSON for input and output data
2. External Communication: The communication between the web application and the external API.
   1. Integration with ChatGPT API
      1. Purpose: To generate AI-based travel recommendations based on user experience
      2. Protocol: HTTPS (REST API)
      3. Data Format: JSON for input and output data (Because we receive the input as the form)
3. Expected Exceptions and Handling
   1. Frontend to Backend communication
      1. Invalid input: if the user submits the invalid data, the backend will return the error code
      2. Session Expiry: if the user’s token expires, the backend will respond with an authentication error.
   2. Backend to External API
      1. API downtime: if the external APIs are unavailable, the backend will notify the user.
   3. Database connection issue
      1. If the MongoDB fails, the backend will notify the user.
4. Security Consideration
   1. Data encryption
      1. All sensitive user data will be encrypted using hash algorithms
      2. Communication between the frontend, backend, and external APIs will be encrypted using HTTPS
   2. API key management
      1. API key for external services will be stored securely
   3. Notification Permission
      1. Users must grant permission to receive notifications and can opt out at any time.

**User Interface Design**

Site Map

The site map defines the hierarchy of pages and their relationships:

Main Page

├── Login Page

└── Registration Page

Logged-In User Dashboard

├── Profile Page

├── Trip Creation Page

├── AI Recommendations Page

├── Expense Management Page

└── Group Management Page

Admin Dashboard

├── User Management

└── Content Moderation

Page Layouts and Navigation Flow

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

**Main Page**

Components:

Logo

Login button

Signup button

Navigation:

Click "Login" → Login Page.

Click "Signup" → Registration Page.

**Login Page**

Components:

Email and password input fields.

"Forgot Password" link.

"Sign Up" link.

Navigation:

Successful login → Logged-In User Dashboard.

Click "Forgot Password" → Password Reset Page (sends email verification).

Click "Signup" → Registration Page.

**Registration Page**

Components:

Email input field.

Button to send email verification code.

Verification code input field.

Password input field.

Profile Fields: Nickname, age, gender.

"Already Have an Account?" Link: Redirects to the Login Page.

"Register" Button.

Navigation:

Click "Register" → Checks the verification code → Redirects to Login Page.

Click "Already Have an Account?" → Redirects to Login Page.

**Password Reset Page**

Components:

Email input field.

Button to send email verification code.

Verification code input field.

New password input field.

"Reset Password" Button.

Navigation:

Click "Reset Password" → Redirects to Login Page.

**Logged-In User Dashboard**

Components:

Sidebar: Navigation links (Profile, Create Trip, My Groups, Expenses, Recommendations).

Main Content Area:

Default: Trip Feed (filterable by destination, date, budget).

Interactive buttons: "Create Trip," "Explore Recommendations," "Start a Group."

Navigation:

Click "Profile" → Profile Page.

Click "Create Trip" → Trip Creation Page.

Click "My Groups" → Group Management Page

Click "Expenses" → Expense Management Page

Click "Recommendations" → AI Recommendations Page

Click a trip post → Trip Details Page (with comments).

**Profile Page**

Components:

Avatar Upload: Drag-and-drop image upload or file selector.

Editable Fields: Nickname, age, gender, description, travel preferences.

"Save Changes" button.

"Discard Changes" button.

"Change Password" link.

Navigation:

Click "Save Changes" → Stays on Profile Page with updated data.

Click "Discard Changes" → Reverts unsaved edits.

Click "Change Password" → Redirects to Password Reset Page.

**Trip Creation Page**

Components:

Form fields: Destination, dates, budget, notes.

"Publish" button.

Navigation:

Click " Publish" → Redirect to Trip Feed with the new post visible.

**AI Recommendations Page**

Components:

Form: Preferences (destination, budget, activities, travel style).

"Generate Recommendations" button.

Results Section: Cards with generated suggestions.

**Expense Management Page**

Components:

Group Selector: choose a trip group.

Expense Log: Table with columns (Category, Amount, Date, Description).

"Add Expense" button → opens a modal form.

Navigation:

Click "Add Expense" → Modal form → Submit updates the table.

**Group Management Page**

Components:

Group List: Displays groups.

Group Creation Form: Filters (destination, budget, preference).

Invitations Section: Pending invites with "Accept/Reject" buttons.

Navigation:

Click a group → Group Chatroom.

**Admin Dashboard**

Components:

Tabs: "Users", "Posts"

Users Tab: Search bar, ban/edit buttons.

Posts Tab: Flagged posts with "Delete" or "Approve" options.

Navigation:

Click "Edit User" → User Edit Page (admin-only fields).

**Accessibility Considerations**

Form Validation:

Clear error messages (e.g., "Invalid email format").

Responsive Design:

Mobile-friendly layouts.

**Component Design**

1. AuthWrapper

1.1 Login

--Responsibilities:

Manage 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 ‘Forget Password’ button.

3. Fetch and store user details on successful login.

1.2 ForgetPassword

--Responsibilities:

Manage the situation that users forget their password.

Send authentication code to email.

--Inputs:

email account

--Outputs:

Authentication code

--Algorithm:

1. Check if the user’s email is authenticated.

2. If so, redirect the user to the password reset page.

3. Fetch and store user’s new password.

2. Travel Group Components

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.

2.2 GroupCard

--Responsibilities:

Displays summary details of a group.

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’ componets for each group.

2.3 GroupDetails

--Responsibilities:

Displays all details of a group.

Show itinerary, members and chat.

--Inputs:

GroupID

--Outputs:

Rendered group details page.

--Algorithm:

1. Fetch group details from API using GroupID.

2. Show group info, itinerary, members and chat.

3. Chat & Messaging Components

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.

3.2 ChatMessage

--Responsibilities:

Displays individual chat messages

Shows sender info and timestamps.

--Inputs:

Message {sender, text, timestamp}

--Outputs:

Rendered chat bubble.

4. Bargaining & Deals Components

4.1. DealList

--Responsibilities:

Displays travel deals available for negotiation.

Filter deals based on preferences.

--Inputs:

Deals[].

--Outputs:

Rendered list of deals

4.2 NegotiateForm

--Responsibilities:

Allow users to negotiate travel deals.

--Inputs:

DealID

Offerprice

--Outputs:

OnNegotiate(dealID, offerPrice).

--Algorithm:

1. Collect user’s offer price.

2. Send negotiation request to backend.

3. Show response (accepted / rejected).

5. Trip Planning Components

5.1. ItineraryBuilder

--Responsibilities:

Enables collaborative trip planning.

Allows group members to vote on activities.

--Inputs:

GroupID

Itinerary[]

--Outputs:

Updated itinerary list.

--Algorithm

1. Fetch existing itinerary.

2. Allow users to add/edit activities.

3. Sync changes in real-time.

6. Review & Rating Components

6.1 ReviewList

--Responsibilities:

Show users reviews for trips

--Inputs:

Reviews[] (String array)

--Outputs:

Rendered list of reviews in text.

6.2 Rating stars

--Responsibilities:

Show users reviews for trips

--Inputs:

Reviews[] (String array)

--Outputs:

OnRate(rating)

**Assumptions**

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.

1. 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.

1. User Environment Assumptions

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