Where Have You Been (Personal Travel Journal)

EE547: Applied and Cloud Computing, Final Project Report

Anni Li (annili@usc.edu), Zeli Liu (zeliliu@usc.edu), Kaisen Ye (kaisenye@usc.edu)
University of Southern California

December 14th, 2024

1. Introduction

In an increasingly interconnected world, people are eager to review and explore their travel experiences. Platforms like Goodreads and RateYourMusic allow users to log books and music, but few tools exist specifically for tracking travel. *Where Have You Been* fills this gap by enabling users to mark visited places, maintain wishlists, and receive timely notifications based on their preferences.

We developed a website with the following features:

- Users can search for and record places they have visited or wish to visit.
- Secure login and signup functionality.
- Visualized analysis of travel history and wishlist data.
- Timely reports based on user preferences.
- Fully deployed on AWS services.

2. Architecture

The proposed system architecture integrates a React-based frontend and a Nodejs-powered backend to enable users to record, analyze, and visualize their travel data. The frontend is responsible for user interactions such as signup, place searches, record updates, and map-based visualizations. Key frontend components include modules for analysis, history, wishlist, user profile, and reports. The backend manages request authentication, data processing and storage.

2.1 Backend

The backend is built using Node.js with AWS Lambda for its scalability and deployment-friendly. We use AWS API Gateway to expose APIs to our frontend. AWS EventBridge is used to trigger scheduled tasks.

2.2 Frontend

The frontend is developed with React and deployed on AWS Amplify. It includes modules for analysis, history, wishlist, user profiles, and reporting.

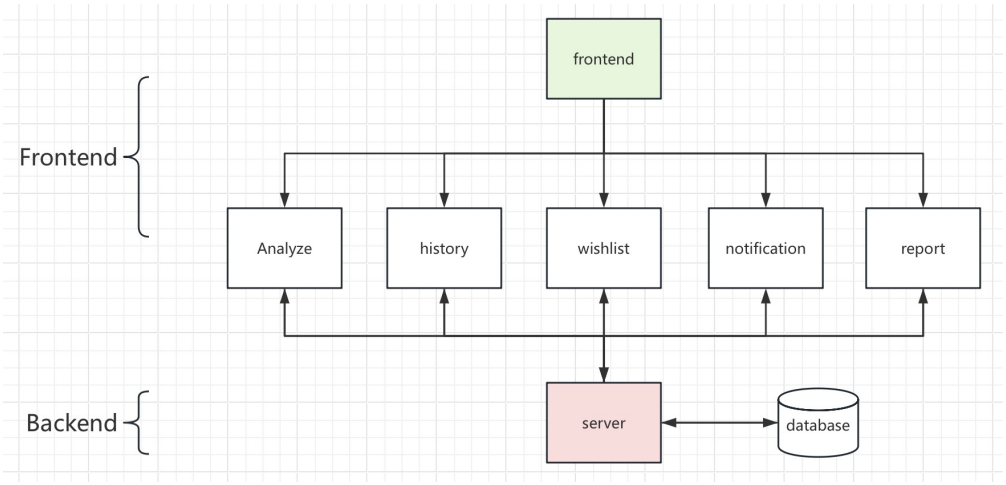

Figure 1: Overall architecture

2.3 Database

We use AWS DynamoDB as our database to manage data efficiently. Below, we explicitly describe the database schema, the approximate structure of each table, and its relationship to client fields, along with field-level validation and decorators.

- **User Table:** Stores information related to the users of the application.
  - `user_id (String, PK)`: Unique identifier for each user. Validator: UUID format.
  - `user_name (String)`: Name of the user. Validator: Alphanumeric
  - `password (String)`: Encrypted password stored as a hash. Validator: Non-empty.
  - `email (String)`: User's email address.
  - `avatar_url (String)`: S3 URL for user avatar img.
  - `createTime (String)`: Timestamp when data is created. Validator: ISO 8601 date format.
  - `modifyTime (String)`: Timestamp when data is updated. Validator: ISO 8601 date format.
- **User_Key Table:** Links users to generated keys for application access.
  - `user_id (String, PK)`: Foreign key referencing the `User Table`.
  - `user_key (String, SK)`: Generated API key for the user. Validator: Alphanumeric.
  - `validate_date (String)`: Expiration date for the key. Validator: ISO 8601 date format.
- **History Table:** Tracks the places users have visited.
  - `user_id (String, PK)`: Referencing the `User Table`.
  - `place_id (String, SK)`: Identifier for the place in GoogleMap. Validator: Alphanumeric.
  - `placeName (String)`: Name for the place.
  - `description (String)`: User's description for the place.
  - `S3_URL (String)`: URL for place img.
  - `createTime (String)`
  - `modifyTime (String)`
- **Wishlist Table:** Stores information about places users wish to visit.

  - `user_id (String, PK)`: Referencing the `User Table`.
  - `place_id (String, SK)`: Identifier for the place in GoogleMap. Validator: Alphanumeric.
  - `createTime (String)`
  - `modifyTime (String)`

- **Report_Subscription Table:** Tracks user preferences for subscriptions.
  - `user_id (String, PK)`: Foreign key referencing the `User Table`.
  - `report_frequency (String)`: Frequency of subscription (e.g., 1, 2, 3, 4). Validator: Enum.
  - `report_type (String)`: Type of subscription (e.g., 1, 2). Validator: Enum.

- **Report Table:** Stores data about reports generated for users.
  - `report_id (String, PK)`: Unique identifier for each report. Validator: UUID format.
  - `user_id (String)`: Referencing the `User Table`.
  - `report_data (String)`: The actual report data.
  - `startTime (String)`: The start date of report Validator: ISO 8601 date format.
  - `endTime (String)`: The end date of report. Validator: ISO 8601 date format.
  - `createTime (String)`: The timestamp report is generated. Validator: ISO 8601 date format.

The above schema ensures data consistency and security through strict validators and decorators applied at the API layer.

2.4 Others

We use AWS S3 for image storage, and AWS Parameter Store for some sensitive configurations, like the GoogleMapAPIKey in our project. In our development, we use Postman for API test and Git for version control.

3. Backend

3.1 User Server

The user server manages login, signup, and authentication while ensuring security and a smooth user experience. Upon a successful login or signup, the server generates a one-day-duration user_key linked to the user's user_id. This key is stored in the User_Key Table and sent to the frontend, where it is saved as a browser cookie, allowing users to stay logged in without repeated authentication.

For each request to our main server, the user_key is required to be contained in the request header and the Authentication Server validates the user_key against the database. If valid, the server retrieves the corresponding user_id and processes the request. If user_key is invalid or missed, our frontend will redirect to the login page.

The Login/Signup Server verifies password against the User Table, where hashed passwords are stored. Once validated, a new user_key is issued. To maintain security, user_id is never exposed to the frontend, and passwords are securely hashed by bcryptjs dependency. This design ensures efficient authentication and user privacy.

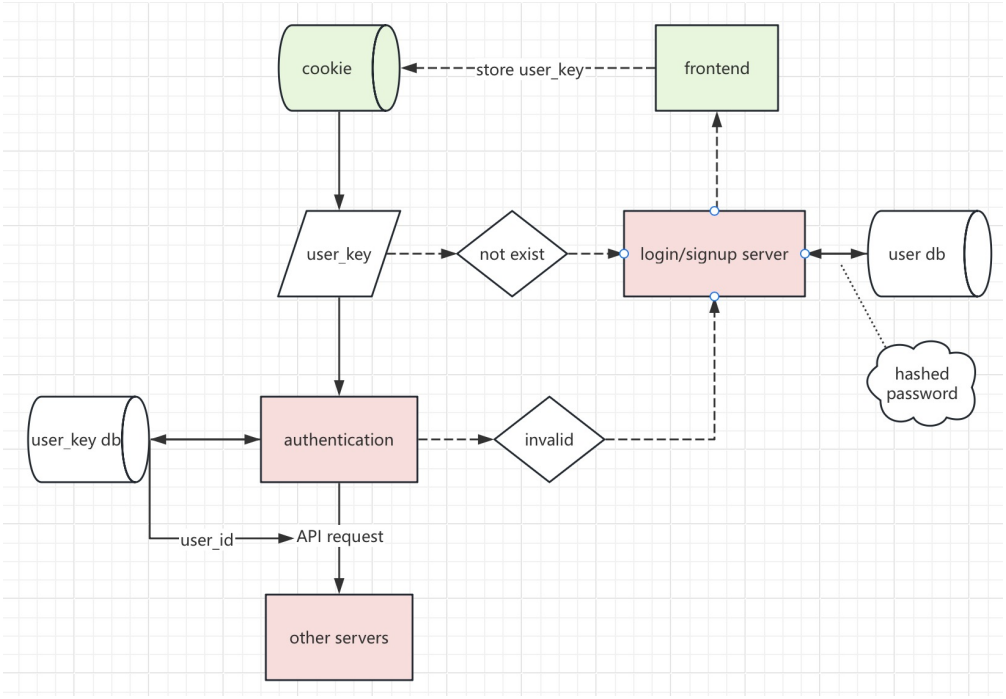

Figure 2: User Server Flow

3.2 Place and Wishlist Server

The system handles place search, image storage, and user wishlist/history management through a combination of Google Maps APIs, AWS S3, and internal servers. When a user searches for a place, the Place Server communicates with the Google Map Text Search API to retrieve relevant place details, such as location information and metadata. The Place Server also fetches the corresponding image as default photo using the Google Map Photo API.

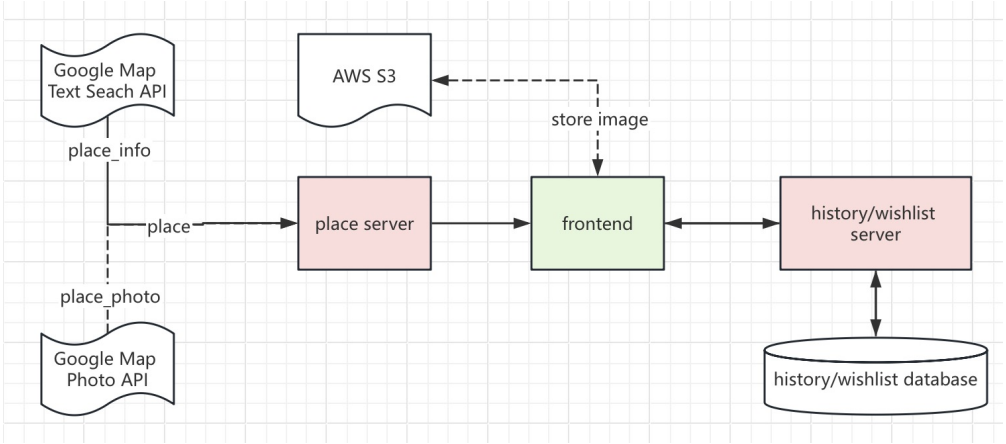

Figure 3: Main Server Flow

The Place Server then sends the place details and default image to the Frontend. Users can replace the default photo with their own uploaded photo, which will be stored in S3.

The History/Wishlist Server manages these user actions by updating the corresponding records in the History/Wishlist Database. The user_id is set to be the primary key of the database, which ensures O(1) query

complexity even if the amount of users go big. The design integrates seamless communication between the Place Server, Frontend, and History/Wishlist Server to provide an efficient and consistent user experience for tracking travel activities.

3.3 Report and Analyze Server

The notification and reporting system is designed to generate periodic user reports, maintain subscription preferences, and provide dashboard analytics seamlessly. The flow integrates frontend interactions, backend servers, and AWS EventBridge for task scheduling.

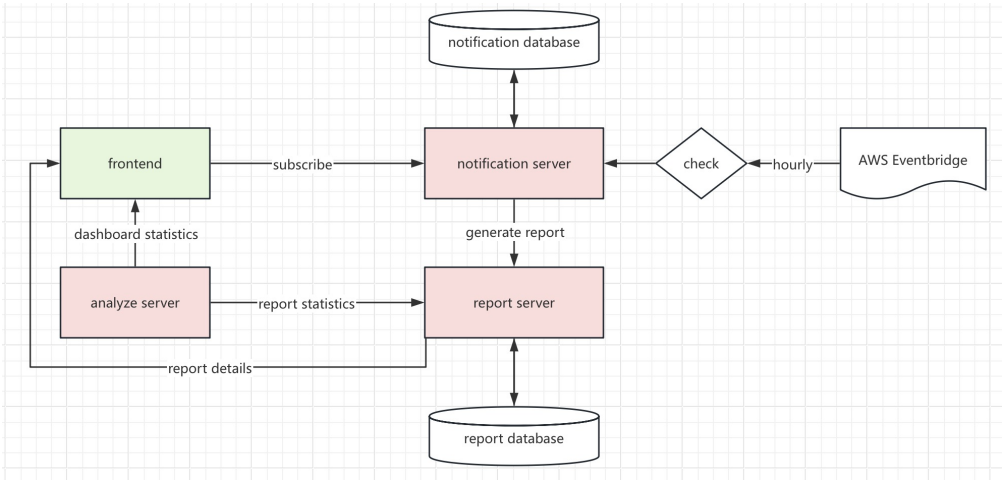

Figure 4: Report Flow

The analyze server uses serval functions from our history/wishlist servers , and computes statistics for frontend dashboards and our report server.

Subscription preferences are stored in the Report_Subscription Table. To ensure timely notifications, AWS EventBridge triggers the notification server hourly, checking for due subscriptions. If any subscriptions require a report, the notification send a request to the report server.

The report server will leverage the Analyze server to retrieve correponding analysis statistics, then generate a static report, and store it into Report Table. The report server provides the frontend existing report list, as well as detailed report content.

4. Frontend

4.1 Dashboard

The Dashboard of Where Have You Been provides users with an intuitive and visually engaging experience to explore and track their travel history. At the top, users can view key metrics, including the total number of places visited, total countries explored, their most visited country, and their favorite type of location, offering a quick summary of their travel activity. A prominent progress indicator displays their achievement toward a 2024 travel target, such as completing 75% of their goal, motivating users to set and track their travel aspirations.

The dashboard further breaks down travel data through visualizations. A pie chart categorizes the types of places visited, such as "Establishment" or "Tourist Attraction," giving users insights into their preferences. A ranked list organizes places by country, providing a detailed summary of user activity across different regions. To enhance the experience, a world map visually highlights the countries visited, with color intensity

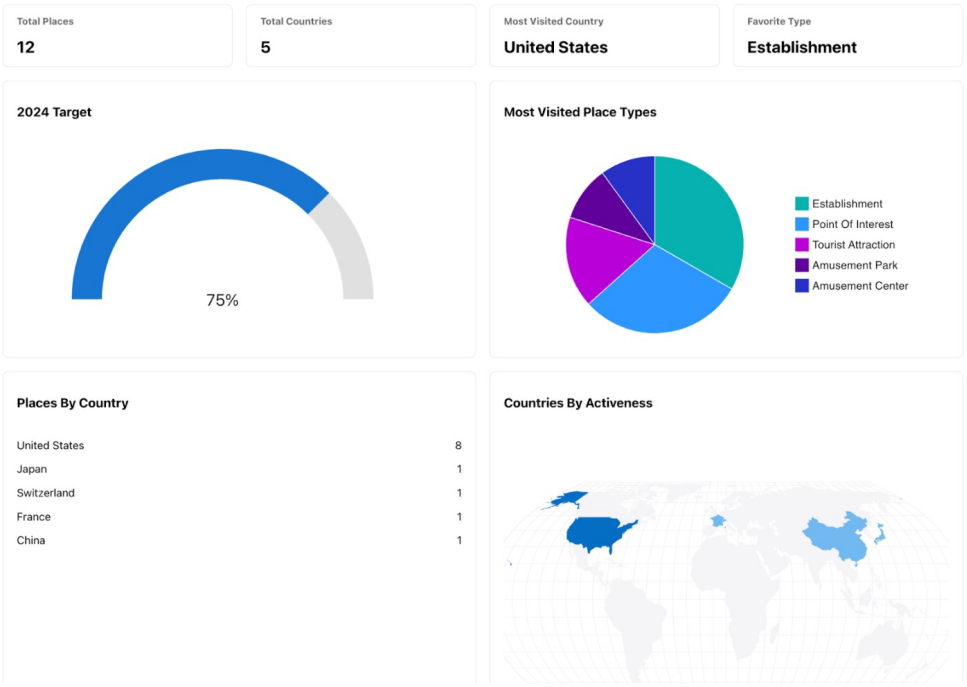

Figure 5: Dashboard Page

representing the level of activity, offering a global perspective of their travel footprint. Together, these features create a seamless and insightful interface, enabling users to reflect on their journeys and plan future trips effectively.

4.2 Travel History Page

The Travel History page provides users with an organized and interactive interface for managing and viewing their past trips. Each travel entry is displayed as a card containing the destination name, associated image, description, rating, and metadata such as the date and location. The cards are visually appealing and allow users to quickly glance at their visited places. Each card also features an edit button, enabling users to update details such as title, description, or rating for a specific location.

At the top of the page, there is an "Add" button, which opens a modal for users to add new travel entries. The modal includes fields for specifying the location (with search functionality), uploading an image, and providing details such as title, description, and rating. This page is designed to be user-friendly, allowing seamless interaction while visually presenting a user's travel history in a structured format.

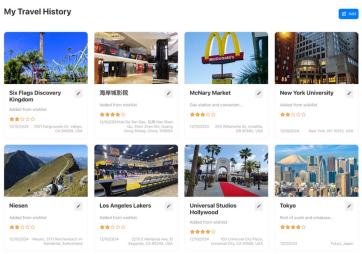

Figure 6: Travel History Page

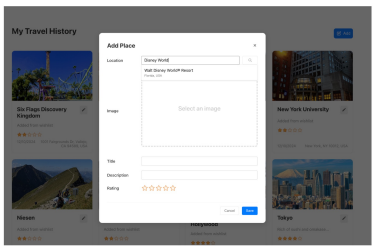

Figure 7: Add Place Window

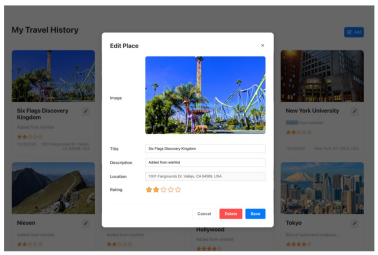

Figure 8: Edit Place Window

The Edit Place modal below provides users with a straightforward interface to modify the details of a previously recorded travel destination. This modal appears when the user selects the "Edit" button on a specific travel entry in their history. It displays the current details of the selected destination, including the image, title, description, location, and rating. Users can easily update any of these fields, such as changing the title, revising the description, or adjusting the rating. The modal also includes a "Delete" button, allowing users to remove the entry entirely from their travel history, and a "Save" button to confirm and save the changes.

4.3 Travel Wishlist Page

The Travel Wishlist page allows users to manage and track locations they aspire to visit. Each wishlist item is displayed as a card, containing details such as the destination's name, address, and relevant categories (e.g., "University," "Coffee Shop"). This structured layout provides a clear and organized overview of potential travel plans.

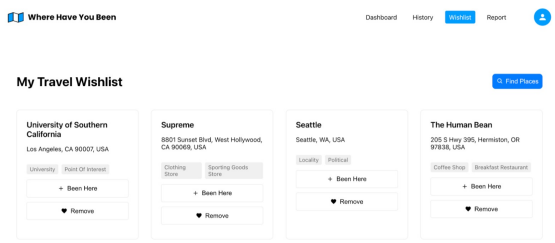

Figure 9: Wishlist Page

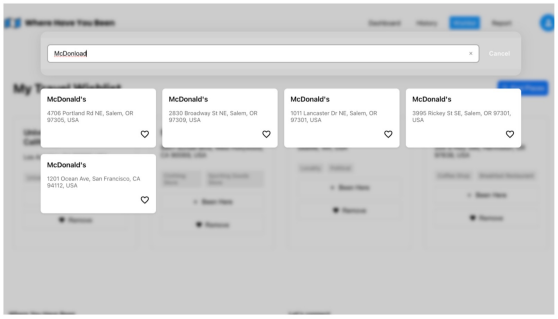

Figure 10: Find Place Window in Wishlist Page

Users can interact with each card through two main actions:

- **Been Here Button**: Marks the location as visited, transferring it from the wishlist to the travel history.
- **Remove Button**: Deletes the location from the wishlist, enabling users to keep their list concise and relevant.

Additionally, a prominent "Find Places" button at the top-right corner provides users with quick access to a location search feature, helping them discover and add new destinations.

The Place Search feature enables users to find specific locations to add to their wishlist or travel history. When a user types a keyword, such as "McDonald's," into the search bar, the system dynamically generates a list of relevant results. Each result is presented as a card, displaying the place name, address, and an interactive heart icon for users to quickly add the location to their wishlist.

4.4 Profile Page

The Profile Settings page provides users with a simple and clear interface to manage their account details and personalize their profile. This straightforward layout ensures that users can quickly access and modify their account settings while maintaining a clean and user-friendly experience. The inclusion of logical sections and a visually distinct "Danger Zone" ensures ease of use and reduces the risk of accidental actions.

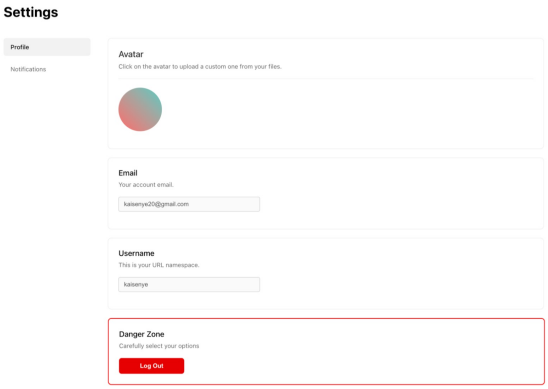

Figure 11: User Profile Setting Page

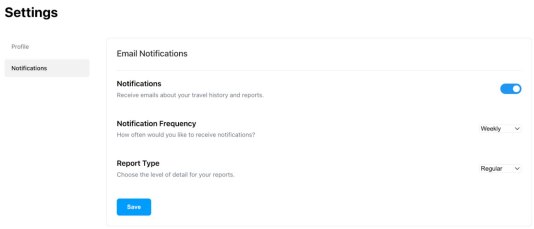

Figure 12: Notifications Setting Page

4.5 Report Page

The Report Page offers a list of generated reports. Users can click on individual reports to open the Report Details Window. The detail window provides users with a summary of their travel activities, including the total places visited, top-rated countries, and states. The Report Details window displays the time range, visit counts, and ratings for destinations. A reminder section highlights pending wishlist items, encouraging users to continue exploring. This organized interface offers a clear and concise overview of travel insights for reflection and future planning.

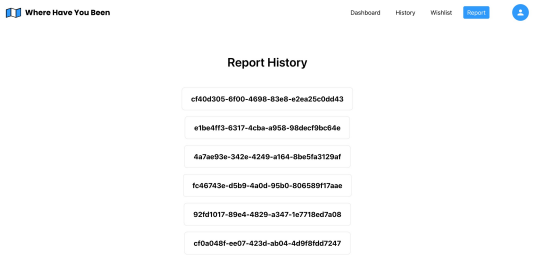

Figure 13: Report Page

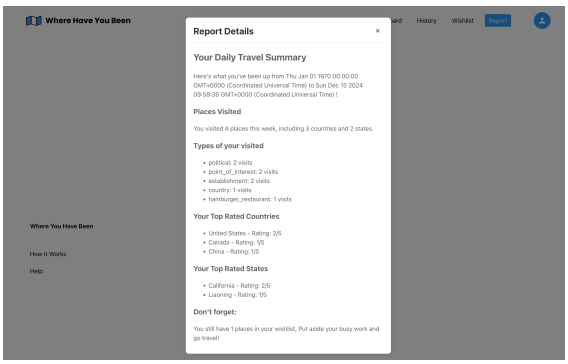

Figure 14: Report Details Window

The Notifications Settings page, as shown below, allows users to configure their preferences for receiving updates about their travel history and reports. This page is designed to enhance user engagement while respecting individual preferences for updates and reports. The clean and intuitive interface ensures users can easily adjust their settings to maintain control over how they interact with the app's notifications.

5. Conclusion and Challenges

The project successfully achieved main goals in our revised proposal, delivering a fully functional personal travel journal with secure authentication, dynamic analysis and visualizations, and user-friendly interactions. Key lessons included architecture design, AWS deployment, data analyze and team collaboration.

With respect to the limited time, some ongoing challenges and potential improvements include:

- Our system was expected to deliver timely notifications to users' email addresses. However, AWS SES (Simple Email Service) requires verified sender domain, which hindered us to implement the real

delivering feature. In the future, we may try to find other email delivering supports to realize this feature.

- One of our planned but not implemented feature is the recommendation system, which can analyze user history to provide more insightful summaries and recommend other visit places. In the future, this feature can be implemented using pre-trained large language model APIs for a smoother user experience.
- Currently, our project queries places search using Google Map API without any pre-processing of user prompts. In the future, the search accuracy can be enhanced by adding some prompt-processing functions, such as sorting search results based on user location, filter search by categories, etc, and enable more advanced search using natural language processing (NLP).

A. References

During development of our project, we referred to some documentation and previous work, including:

1. **Google Maps Platform APIs**
   Google Developers. (n.d.). *APIs by platform.* Retrieved December 15, 2024, from `https://developers.google.com/maps/apis-by-platform`
2. **Visualization Apps for Travel**
   Christensen, C. (n.d.). *6 cool apps to visualize your travels.* LinkedIn. Retrieved December 15, 2024, from `https://www.linkedin.com/pulse/6-cool-apps-visualize-your-travels-chip-christensen/`
3. **Travel Planner App**
   Ahmadluay9. (n.d.). *Travel planner app.* GitHub. Retrieved December 15, 2024, from `https://github.com/ahmadluay9/travel-planner-app?tab=readme-ov-file`

B. Timeline

Nov. 15	General structure, function, and outline planned
Nov. 18	Frontend basics constructed
Nov. 27	Login/signup, map API completed, deployment setup
Nov. 30	Frontend-backend integration finished
Dec. 7	Notifications, analysis, and reporting completed
Dec. 8-11	Bug fixes and frontend refinement

C. Contributions

Kaisen	Frontend development, design, and deployment
Zeli	Backend infrastructure, deployment, and architecture design
Anni	Analysis, notification, and report generation

D. Appendix: Project Links

The following are the project resources, including the frontend application link and the GitHub repositories for both frontend and backend code.

Resource	Link
Frontend Application	https://main.dv93zam2h6wfe.amplifyapp.com/
Frontend GitHub Repository	https://github.com/kaisenyeAtUSC/ee547-whereYouHaveBeen-frontend.git
Backend GitHub Repository	https://github.com/allenLau0708/ee547_final_whereHaveYouBeen.git

E. API Documentation

The following provides an overview of the API endpoints used in the project. Detailed request methods, query parameters, sample payloads, and responses are documented in Postman and available on GitHub.

Note: Endpoint security includes user authentication, where a **user_key** is required in request headers. Complete documentation, including query parameters, and error handling, is available on GitHub.

User API

Endpoint	`/users`
`/`	Retrieves the user profile (GET)
`/signup, /login`	Handles user signup and login (POST)

Place API

Endpoint	`/places`
`/`	Searches for places using text input (GET)
`/:placeID`	Retrieves place details based on the provided `placeID` (GET)

History and Wishlist API

Endpoint	`/history` and `/wishlist`
`/`	Retrieves or adds entries (GET/POST)
`/:placeID`	Deletes a specific entry using `placeID` (DELETE)

Note: The wishlist API operates similarly to the history API, sharing the same structure and functionality.

Notification API

Endpoint	`/notification`
`/`	Retrieves a list of available reports (GET)
`/:reportID`	Fetches details of a specific report (GET)
`/subscription`	Manages report subscriptions (GET/POST/DELETE)

Analysis API

Endpoint	`/analyze`
`/`	Retrieves dashboard statistics and analytics (GET)

F. Project Dependencies

The following dependencies were used in this project, as specified in the `package.json` file:

- `@aws-sdk/client-dynamodb`: DynamoDB client library for AWS SDK (`3.699.0`).
- `@aws-sdk/client-lambda`: Lambda client library for AWS SDK (`3.699.0`).
- `@aws-sdk/client-ses`: Simple Email Service (SES) client library for AWS SDK (`3.699.0`).
- `@aws-sdk/client-ssm`: Systems Manager (SSM) client library for AWS SDK (`3.699.0`).
- `@aws-sdk/lib-dynamodb`: Higher-level library for interacting with DynamoDB (`3.699.0`).
- `aws-xray-sdk`: AWS X-Ray SDK for tracing AWS services and APIs (`3.10.2`).
- `axios`: HTTP client for making API requests (`1.7.7`).
- `bcryptjs`: Library for hashing passwords (`2.4.3`).
- `body-parser`: Middleware for parsing HTTP request bodies (`1.20.3`).
- `cors`: Middleware for enabling Cross-Origin Resource Sharing (`2.8.5`).
- `dotenv`: Loads environment variables from a `.env` file (`16.4.5`).
- `express`: Web application framework for Node.js (`4.21.1`).
- `node-fetch`: Lightweight library for making HTTP requests (`2.7.0`).
- `serverless-http`: Middleware for integrating Express with AWS Lambda (`3.2.0`).