Management of Software Development

Group 3

Teacher : Nguyen Nhat Hai

December 2025

Contents

1 Introduction

The purpose of this document is to introduce the development project of a travel–planning web application. The project focuses on designing a simple, user–friendly, and fully functional application implemented entirely in JavaScript, HTML, and CSS. This technological choice was made to allow all team members—including those with little programming experience—to participate effectively in the development process. The application aims to simplify trip organization while providing interactive, personalized, and collaborative features to enhance the travel experience.

1.1 Purpose of the Document

The objective of this document is to present the overall context, scope, and foundational elements of the project. It describes the main goals of the application, defines the terminology used throughout this document, and clarifies the boundaries of the system to be developed. This introduction also serves as a reference for stakeholders to understand the project's intentions and expected outcomes.

1.2 What to build - Product Scope

The application aims to centralize essential travel–planning tools into a single platform. Its main goals include:

- Simplifying trip organization by gathering different key features in one application.
- Offering a visual and interactive experience through maps and itineraries.
- Allowing budget tracking and travel sharing among users with a review section.
- Facilitating teamwork and practicing real project management methods.

To achieve these goals, the application will include the following core functionalities:

- Destination searching and browsing.
- Adding destinations to a favorites list.
- Creating and visualizing itineraries.
- Managing travel budgets.
- Posting and consulting user reviews.
- Managing user profiles and viewing personal history.

- Input validation and secure HTML escaping.
- Integration of an interactive map module.
- Front-end development using JavaScript, HTML, and CSS.
- Use of a simple database for data persistence.
- Debugging and iterative improvements.

1.3 What to Do - Project Scope

- Plan the overall project and define responsibilities.
- Conduct system analysis:
  - Problem statement.
  - Proposed solution.
  - User personas.
  - Market analysis.
- Specify software requirements (SRS):
  - Use case diagram.
  - Functional and non-functional requirements.
- Distribute coding tasks among team members.
- Design the system architecture and user interface.
- Combine code contributions and perform iterative debugging.
- Manage team communication and collaboration.
- Write individual report sections and merge them collaboratively.
- Write the final report summarizing technical and managerial work.
- Review, refine, and validate the final deliverables.

1.4 Out-of-Scope

- Use of backend frameworks or server-side languages.
- Advanced database systems (only a simple storage system is used).
- Real-time collaborative editing or real-time map tracking.
- Deployment of the application as a commercial product.

1.5 Background

Travel planning often requires multiple tools such as map services, budgeting applications, and recommendation platforms. This project aims to consolidate these elements, offering users a simple and intuitive experience. Additionally, the project provides the team with hands-on experience in collaborative software development, project management, and full-stack implementation.

2 System Analysis

2.1 Problem Statement

Travelers frequently rely on multiple tools to plan their trips, such as map services, budgeting applications, review platforms, and destination search engines. This fragmentation forces users to switch constantly between different interfaces, making the planning process inefficient, difficult to organize, and prone to missing important information.

In addition, many existing travel-planning services are either too complex for casual users or too limited to provide a complete overview of a trip. Users often lack a simple way to estimate the cost of staying a certain number of days in a chosen city, to track their travel budget, or to consult reviews in a unified environment.

There is therefore a need for a lightweight, accessible, and intuitive application that centralizes essential travel-planning features. Such an application should allow users to quickly search for destinations, estimate trip costs, manage budgets, create itineraries, save favorites, and access community reviews—all within a single, easy-to-use platform.

2.2 Proposed Solution

The proposed solution is a web-based travel-planning application that consolidates all key trip-organization tools into one unified interface. Built using JavaScript, HTML, and CSS, the application provides a smooth and interactive user experience.

The platform allows users to search and explore destinations, view essential information, and estimate the cost of staying a chosen number of days. Users can add places to a favorites list, build and visualize itineraries through an interactive map, and manage their overall budget using integrated budgeting features. A shared review section enables users to post and consult feedback from others, enriching the decision-making process. Additionally, itineraries can be saved into a personal history, allowing users to easily reload all information related to a previously created trip whenever needed.

By gathering all these functionalities in a single web application, the solution simplifies every stage of trip preparation while remaining intuitive, visually engaging, and accessible to any user looking for a clear and centralized way to plan their travels.

2.3 User Personas

To better understand user needs and guide the design of the application, two representative user personas were defined. These personas reflect typical usage scenarios of the travel-planning platform.

2.3.1 Persona 1: Emma — The Occasional Traveler

- **Age**: 22
- **Status**: Undergraduate student
- **Digital skills**: Basic web application user
- **Travel habits**: Travels a few times per year, mainly for leisure

Objectives Emma aims to organize short trips efficiently without investing too much time in planning. She seeks a quick overview of destinations and estimated costs before making travel decisions.

Challenges Emma finds many existing travel platforms too complex and fragmented. Using multiple tools for maps, budgeting, and reviews makes trip preparation confusing and time-consuming.

Expectations from the system

- Simple and intuitive destination search
- Fast cost estimation based on trip duration
- Easy access to user reviews
- Ability to save favorite destinations

Value provided by the application The proposed application centralizes all essential planning features in a single, user-friendly interface, allowing Emma to plan trips quickly and with minimal effort.

2.3.2 Persona 2: Lucas — The Structured Trip Planner

- **Age**: 30
- **Profession**: Full-time employee
- **Digital skills**: Comfortable with web-based tools
- **Travel habits**: Plans several trips per year, often multi-day stays

Objectives Lucas prefers detailed and well-organized travel plans. He wants to manage budgets, structure itineraries, and maintain control over expenses throughout the planning process.

Challenges Lucas is frustrated by the lack of integration between existing travel tools. Budget tracking and itinerary visualization are often missing or poorly supported.

Expectations from the system

- Budget management and expense tracking
- Interactive itinerary visualization
- Ability to save and reload previous trips
- Centralized access to destination reviews

Value provided by the application By combining itinerary planning, budget control, and community feedback into a single platform, the application enables Lucas to plan complex trips efficiently and reliably.

2.4 Market Analysis

As more people rely on online resources to plan their travels, the digital travel-planning industry has expanded dramatically. Travelers today, especially students, young adults, and those on a tight budget, seek out platforms that can centralize the information and simplify the preparation. But the market is currently very fragmented: itinerary planners, budgeting tools, review platforms, and map services are frequently distinct applications. Because of this fragmentation, users must constantly switch between tools like Google Maps, TripAdvisor, TravelSpend, and Wanderlog, which can complicate and lengthen the planning process.

While current rivals provide cutting-edge features, they also have a number of disadvantages. Although they offer comprehensive itinerary management, apps like TripIt or Wanderlog may seem complicated to novices or infrequent travelers. Others, such as TripAdvisor, lack integrated planning tools but concentrate on recommendations. This complexity highlights a clear opportunity for a lightweight, accessible solution that brings key features together without overwhelming the user.

By providing a straightforward, user-friendly interface based on key features like destination search, favorites, interactive maps, itinerary visualization, budgeting, and user reviews, the suggested application closes this gap. Its emphasis on usability and clarity sets it apart from rivals with more features. The project is still in line with current trends like digital trip planning, personalized travel experiences and is following the global tendency to use more simplified and versatile tool. Furthermore, using a website makes it accessible by all users, with the simple need of a web browser and an internet connection. All things considered, the market context demonstrates the value of a simplified all-in-one tool, particularly for novices, students, and tourists looking for a simple planning process.

3 Software Requirements Specification (SRS)

3.1 Use Case Diagram

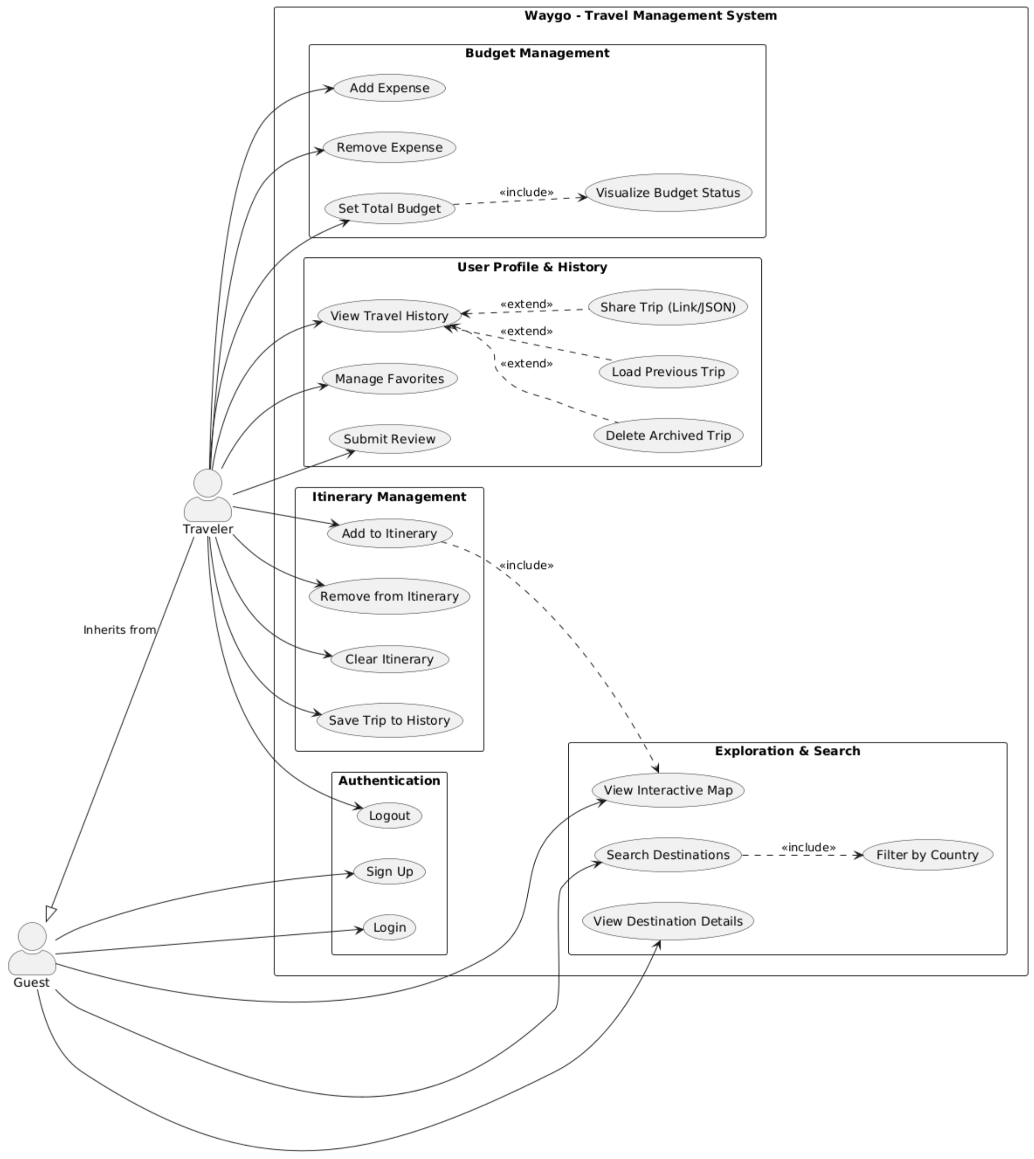


Figure 1: Waygo Use Case

3.2 Functional Requirements

3.2.1 Authentication and User Profile

This code is designed to manage how the user profile is displayed on the website, ensuring the interface adapts depending on whether the user is logged in or not, while also handling the logout process efficiently. In this code there are two functions :

- The applyProfileUI() function dynamically adjusts the user interface based on the user's authentication status. It begins by retrieving any stored data from the browser's localStorage, allowing the system to determine whether a user is currently logged in. Depending on this information, the function updates the profile link to either display the logged-in user's name or redirect to the login page. It also shows or hides the logout button accordingly, ensuring a clear and intuitive interface. This mechanism enhances the user experience by providing a navigation layout that adapts to the user's current status.

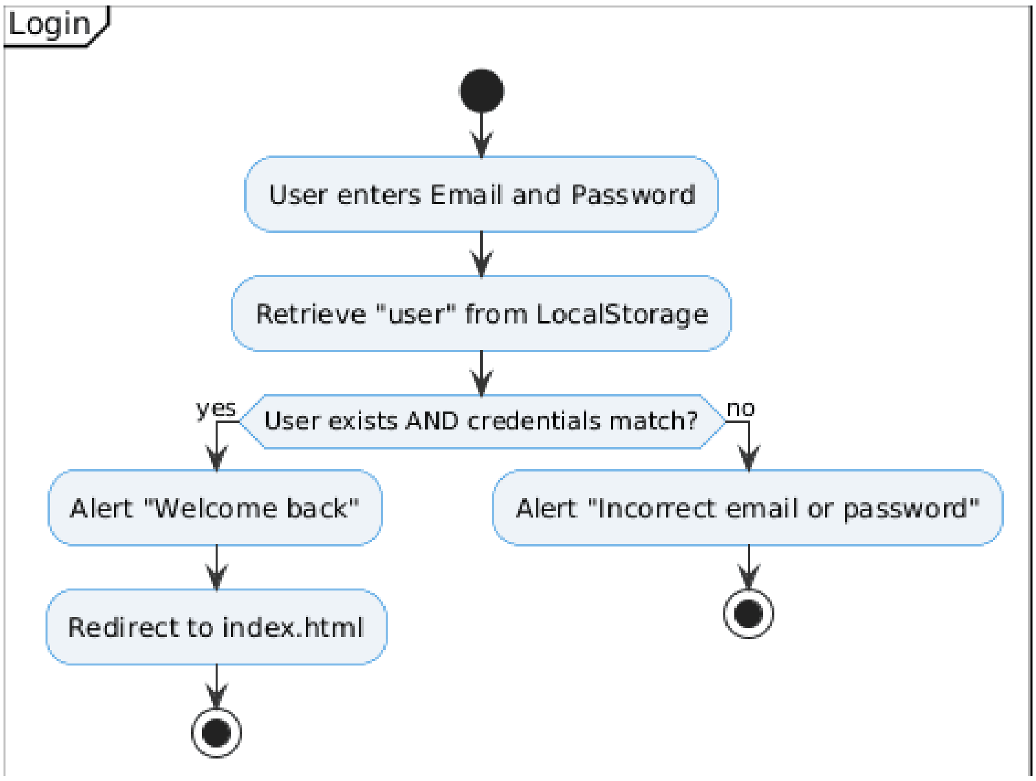

Login
User enters Email and Password
Retrieve "user" from LocalStorage
User exists AND credentials match?
yes
no
Alert "Welcome back"
Alert "Incorrect email or password"
Redirect to index.html

Figure 2: Login Activity Diagram

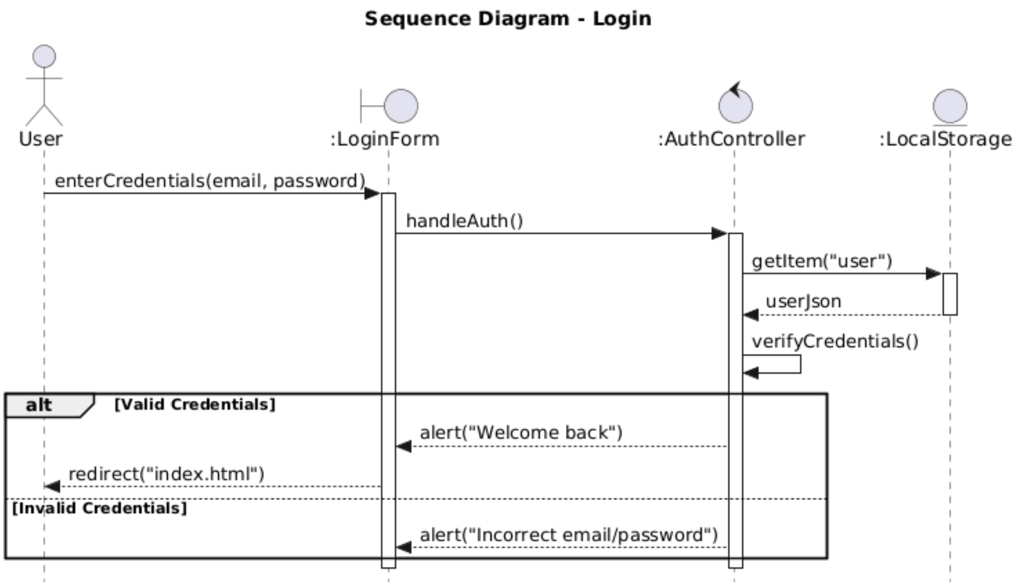


Figure 3: Login Sequence Diagram

- The logout() function manages the entire logout process. It first asks the user for confirmation to prevent accidental disconnection. If confirmed, it removes the stored user data from localStorage, effectively ending the session. The function then calls applyProfileUI() to refresh the interface based on this new state, and finally redirects the user to the login page. This ensures a smooth, secure, and consistent transition between logged-in and logged-out states within the user management system.

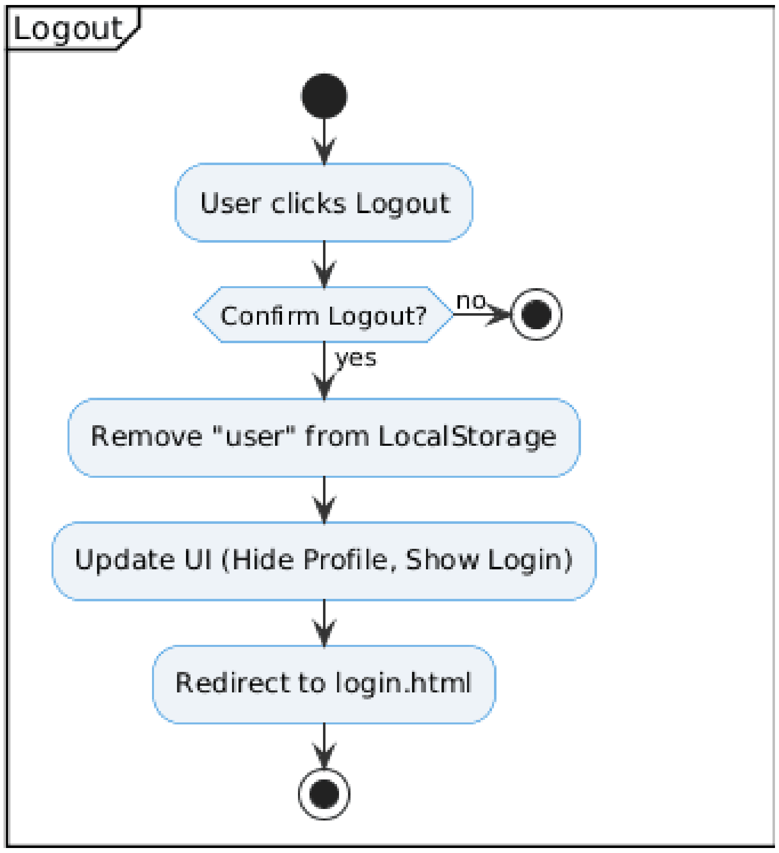

Logout
User clicks Logout
Confirm Logout?
no
yes
Remove "user" from LocalStorage
Update UI (Hide Profile, Show Login)
Redirect to login.html

Figure 4: Logout Activity Diagram

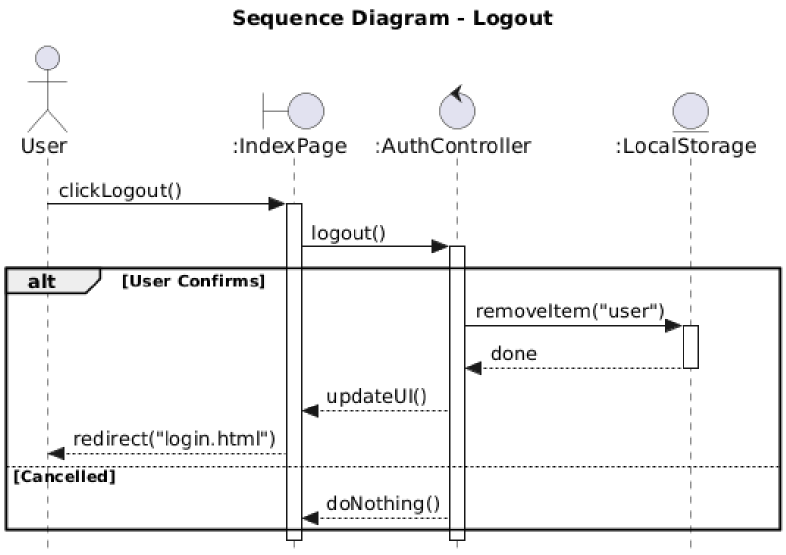


Figure 5: Logout Sequence Diagram

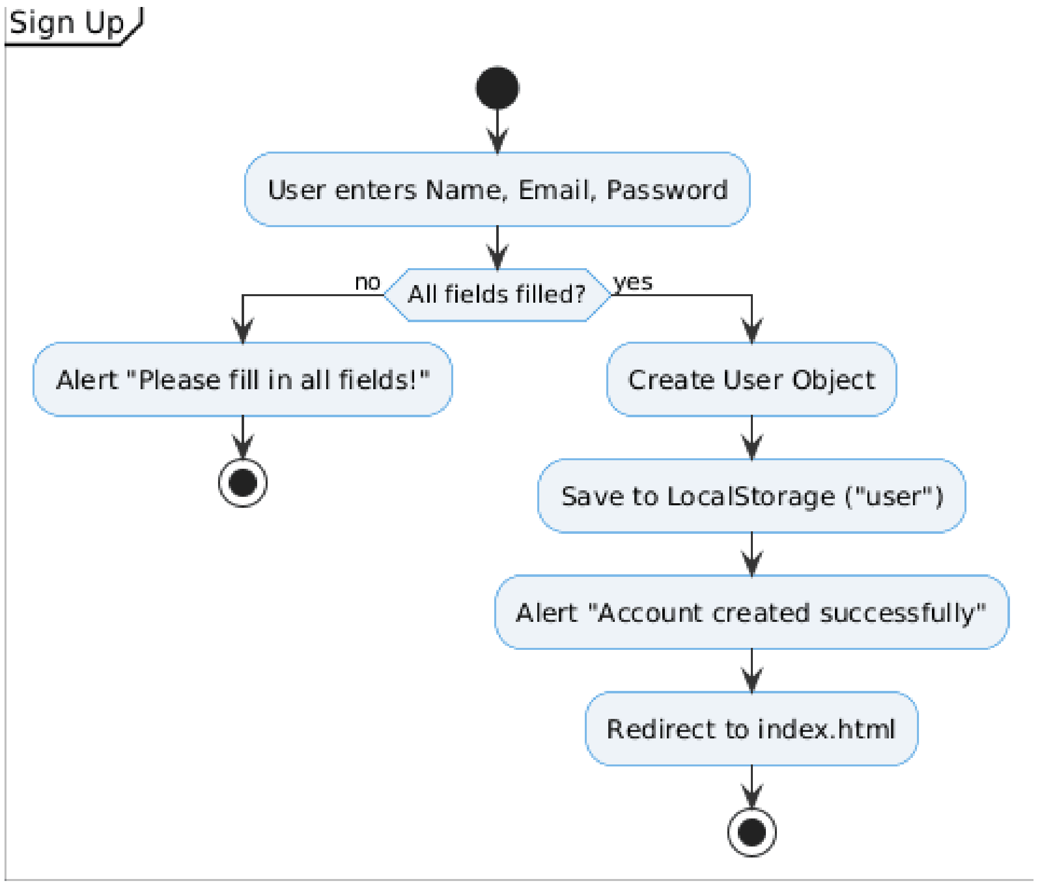

Sign Up
User enters Name, Email, Password
All fields filled?
no
yes
Alert "Please fill in all fields!"
Create User Object
Save to LocalStorage ("user")
Alert "Account created successfully"
Redirect to index.html

Figure 6: Sign Up Activity Diagram

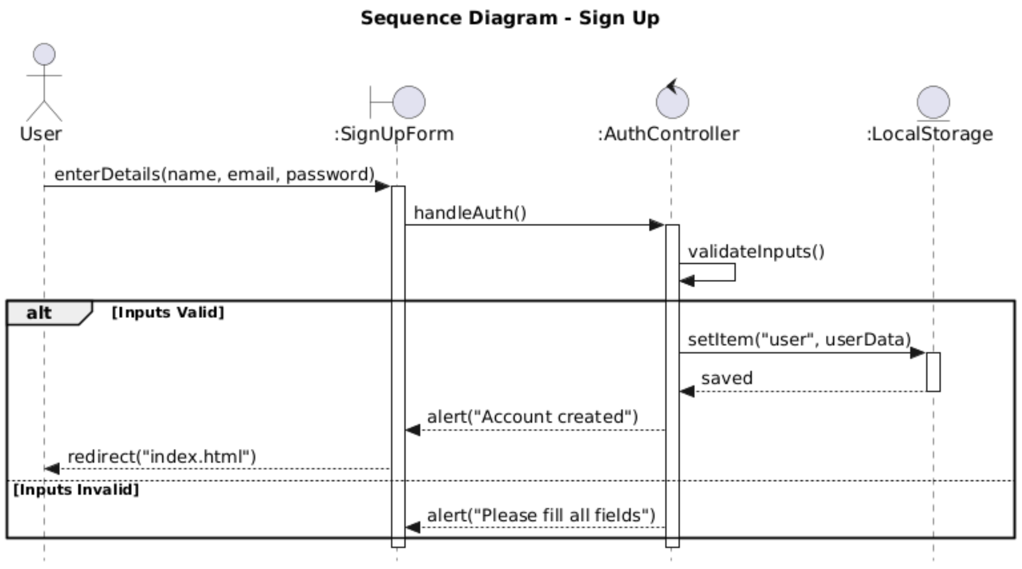


Figure 7: Sign Up Sequence Diagram

3.2.2 Destination Search

FR-SEARCH-01: Populate Country Filter

The system shall dynamically populate the country selection dropdown based on the available destination data to ensure the filter options are always current.

- **Trigger:** The application initializes (execution of `_runWaygoInit`) or the function `populateCountryF` is called explicitly.
- **Pre-condition Validation:** The `destinations` array must be loaded, and the DOM element with ID `filter-country` must exist.
- **Data Processing:**
  - The system shall clear any existing options in the select element and append a default option: "All countries".
  - The system shall extract all unique `country` values from the `destinations` data array.
  - The system shall sort these country names alphabetically before creating the `<option>` elements.
- **Output:** The dropdown menu is populated with the sorted list of unique countries.

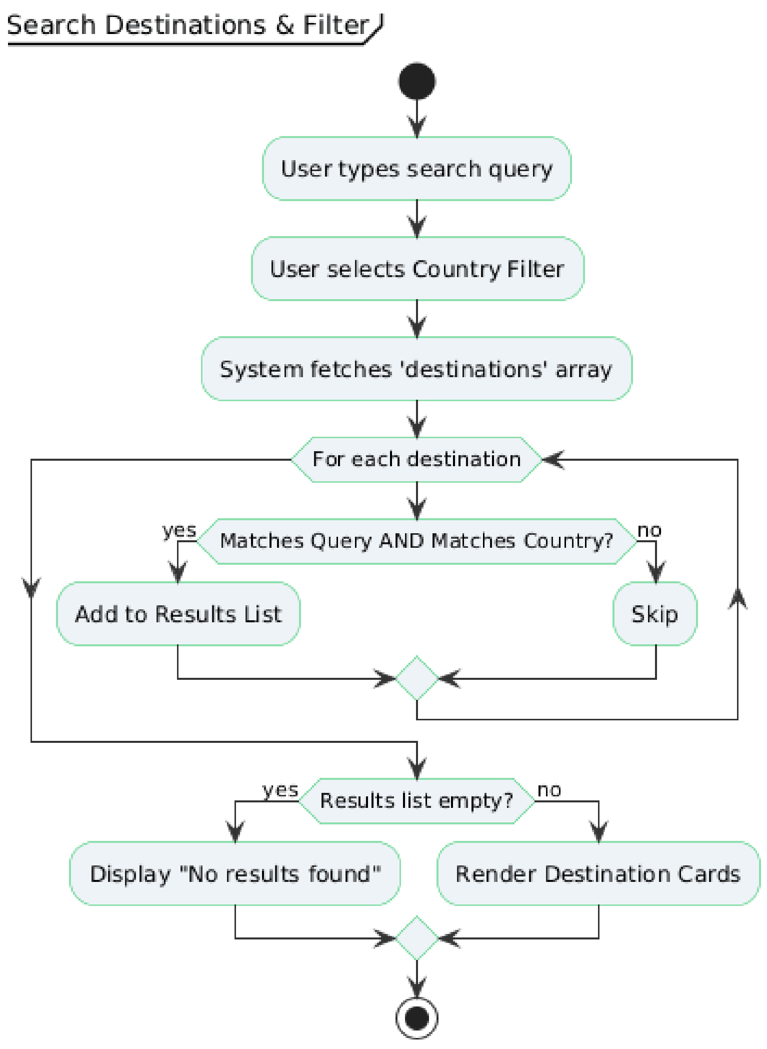

Search Destinations & Filter
User types search query
User selects Country Filter
System fetches 'destinations' array
For each destination
yes
Matches Query AND Matches Country?
no
Add to Results List
Skip
yes
Results list empty?
no
Display "No results found"
Render Destination Cards

Figure 8: Search Destinations and Filter Activity Diagram

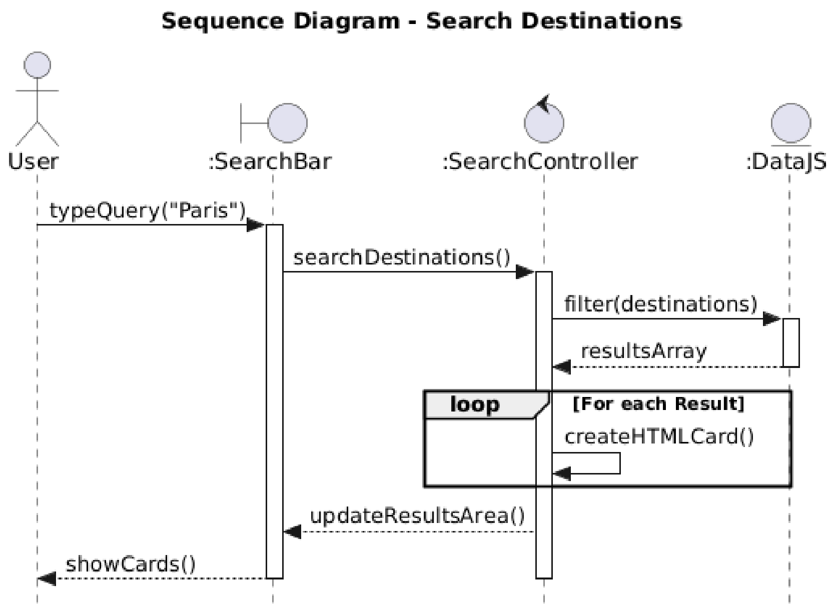


Figure 9: Search Destinations Sequence Diagram

FR-SEARCH-02: Execute Destination Search

The system shall filter the displayed destinations based on user input, matching against multiple data fields and the selected country.

- **Trigger:** The user types in the `search-input` field or selects a value in the `filter-country` dropdown (invoking `searchDestinations()`).
- **User Input:** The system accepts a text string for keywords and a selection for the country filter.
- **Data Processing:**
  - The system shall normalize the search query by trimming whitespace and converting it to lowercase.
  - The system shall apply a logical AND condition: The destination must match the selected country (if one is chosen) AND the keyword must appear in the destination's `name`, `country`, `description`, or `activities`.
- **Output:** An array of `filtered` destination objects is prepared for rendering.

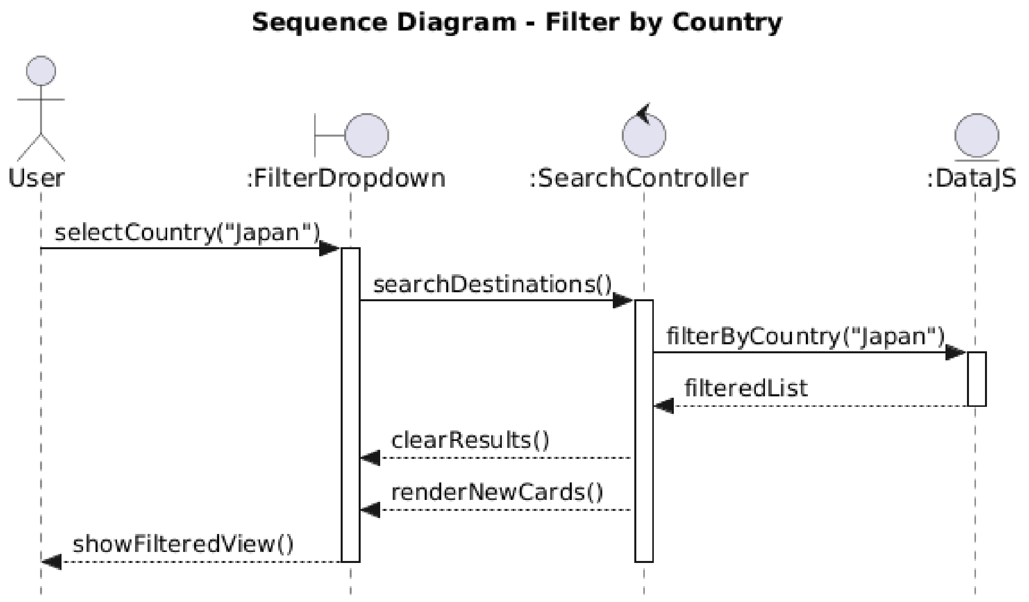


Figure 10: Filter Sequence Diagram

FR-SEARCH-03: Render Search Results

The system shall generate and display visual cards for each destination found during the search process.

- **Trigger:** The filtering process completes within `searchDestinations()`.

- **Pre-condition Validation:** The system must check the length of the `filtered` results. If the length is 0, the system shall display a "No results found" message and abort further rendering.
- **Data Processing:**
  - The system shall generate HTML for each destination, including an image, title, description, and list of activities.
  - The system shall sanitize all dynamic text using `escapeHtml()` to prevent XSS attacks.
  - The system shall attach an `onerror` handler to the image tag to load a fallback SVG if the image source fails.
- **Output:** The `results` container inner HTML is updated with the generated destination cards.

FR-SEARCH-04: Destination Card Actions

The system shall provide interactive buttons within each result card to allow the user to interact with other modules.

- **Trigger:** The user clicks one of the action buttons rendered within a destination card.
- **User Input:** Click events on "Favorite", "Add to itinerary", or "View on map" buttons.
- **Data Processing:**
  - **Favorite:** The system calls `addToFavorites(id)` to save the destination ID to LocalStorage.
  - **Itinerary:** The system calls `addToItinerary(id)` to push the destination to the active itinerary array.
  - **Map:** The system calls `viewOnMap(id)` to pan and zoom the map view to the destination's coordinates.
- **Output:** The respective module (Favorites, Itinerary, or Map) is updated immediately.

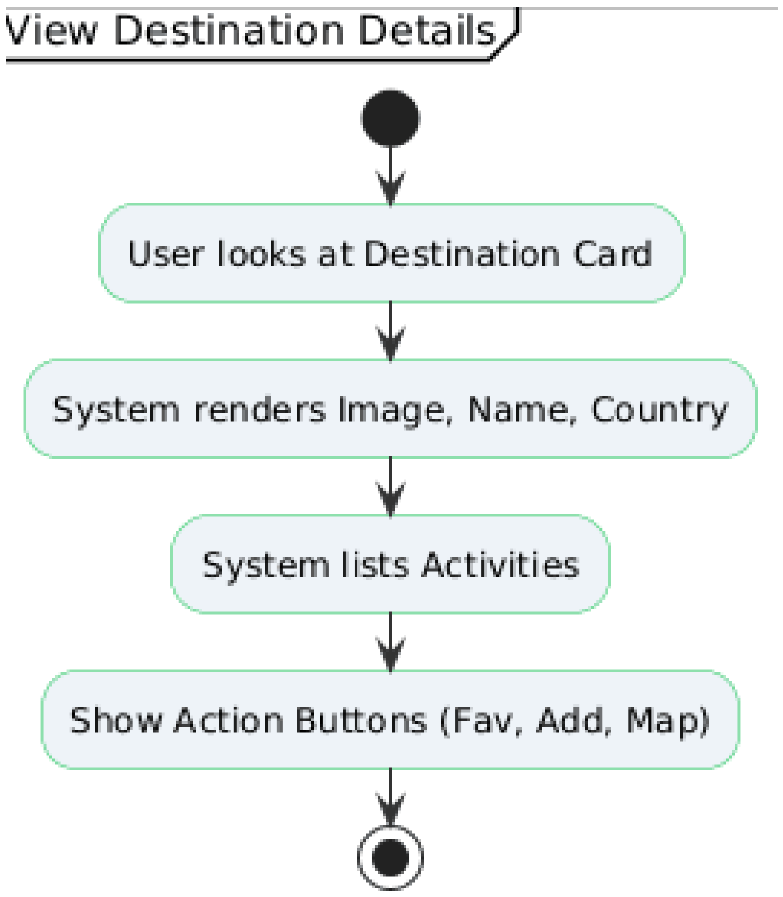

View Destination Details
User looks at Destination Card
System renders Image, Name, Country
System lists Activities
Show Action Buttons (Fav, Add, Map)

Figure 11: View Destination Details Activity Diagram

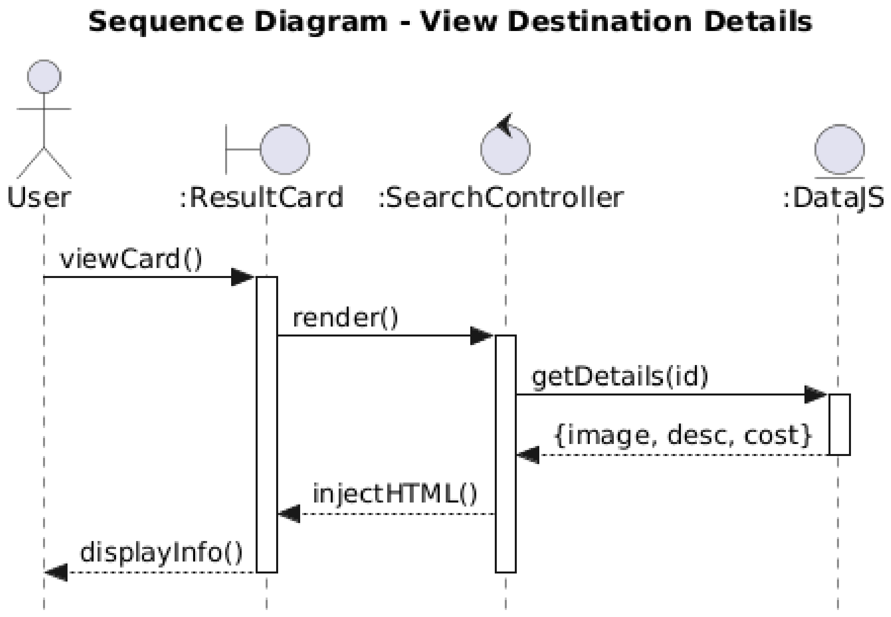


Figure 12: View Destination Details Sequence Diagram

3.2.3 Interactive Map and Itinerary Management

The file itinary.js manages a user's itinerary for a travel application, ensuring the list of destinations is preserved even after the browser is closed by using local storage. The process begins by either loading the saved itinerary or initializing an empty list if no data is found. The main function allows a new destination to be added to the itinerary, provided it is not already present, and if it is added, it immediately updates the storage, the screen display, and the map view to reflect the change. Conversely, another function allows a specific destination to be removed from the itinerary, which also triggers an immediate update of all views. It is also possible to completely clear the itinerary after user confirmation, thereby resetting the trip. For the user-facing part, a dedicated function takes the stored itinerary, retrieves the details for each destination (name, country, cost), calculates the total estimated cost of the trip, and generates a detailed list displayed on the web page, including buttons to either view each destination on the map or remove it from the list.

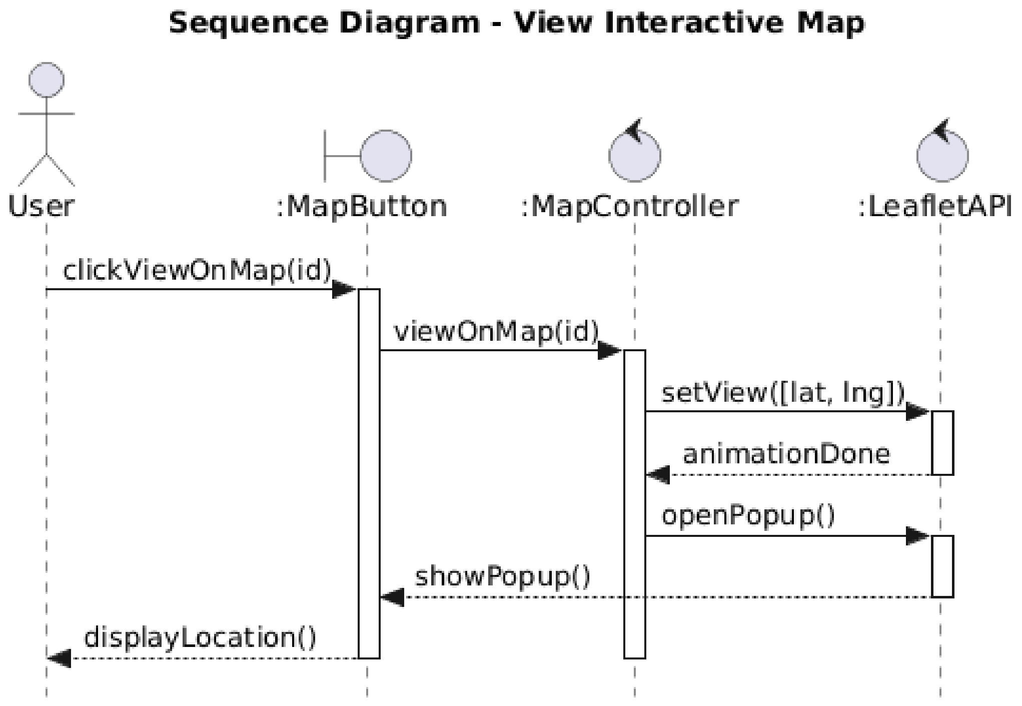


Figure 13: View Interactive Map Sequence Diagram

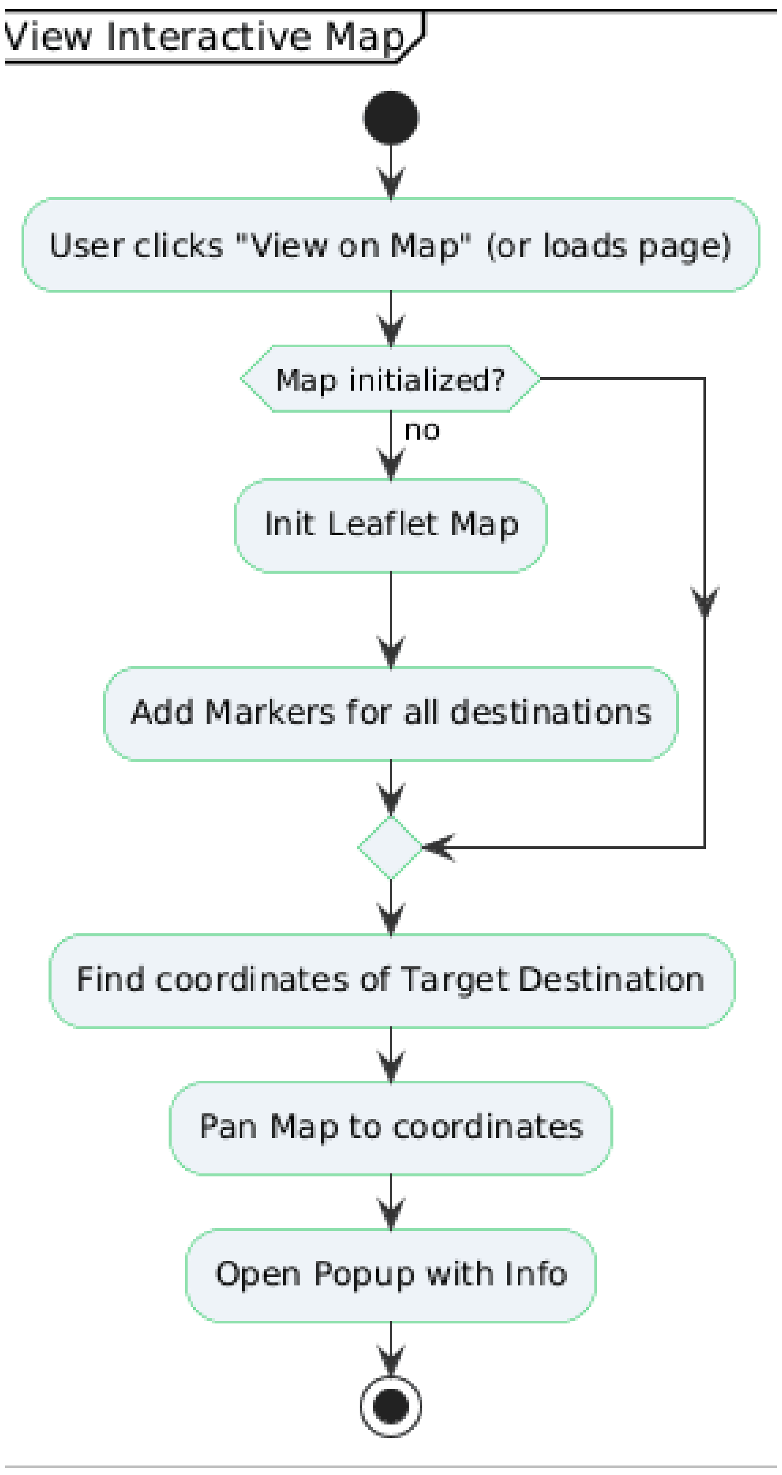

View Interactive Map
User clicks "View on Map" (or loads page)
Map initialized?
no
Init Leaflet Map
Add Markers for all destinations
Find coordinates of Target Destination
Pan Map to coordinates
Open Popup with Info

Figure 14: View Interactive Map Activity Diagram

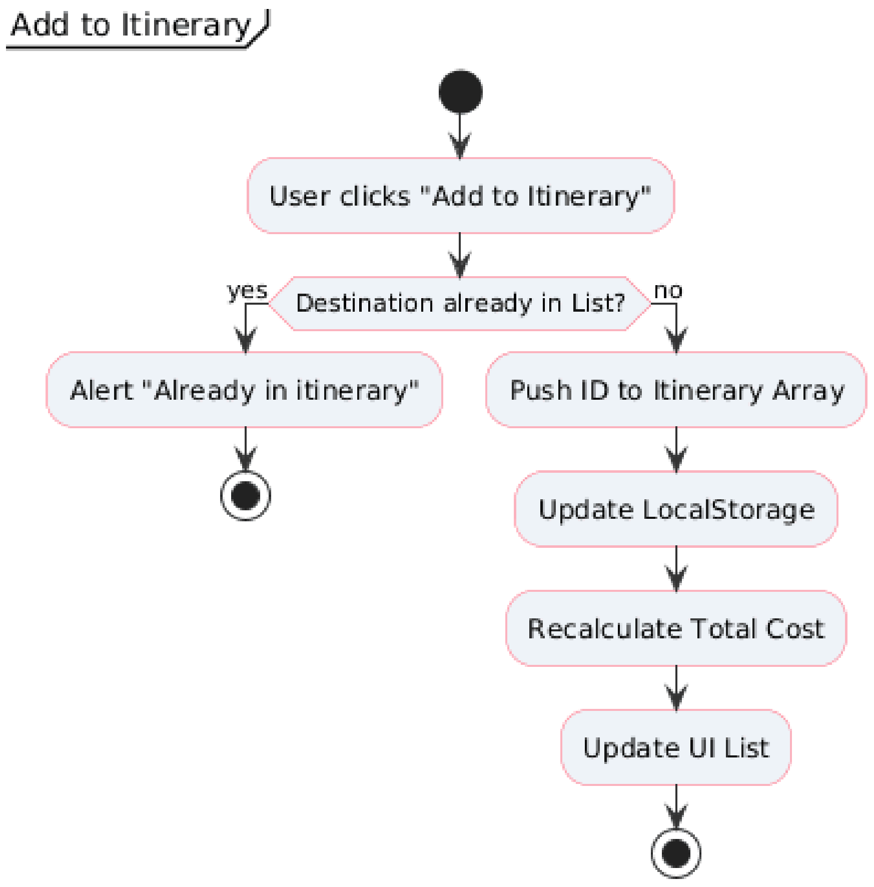

Add to Itinerary
User clicks "Add to Itinerary"
Destination already in List?
yes
no
Alert "Already in itinerary"
Push ID to Itinerary Array
Update LocalStorage
Recalculate Total Cost
Update UI List

Figure 15: Add Itinerary Activity Diagram

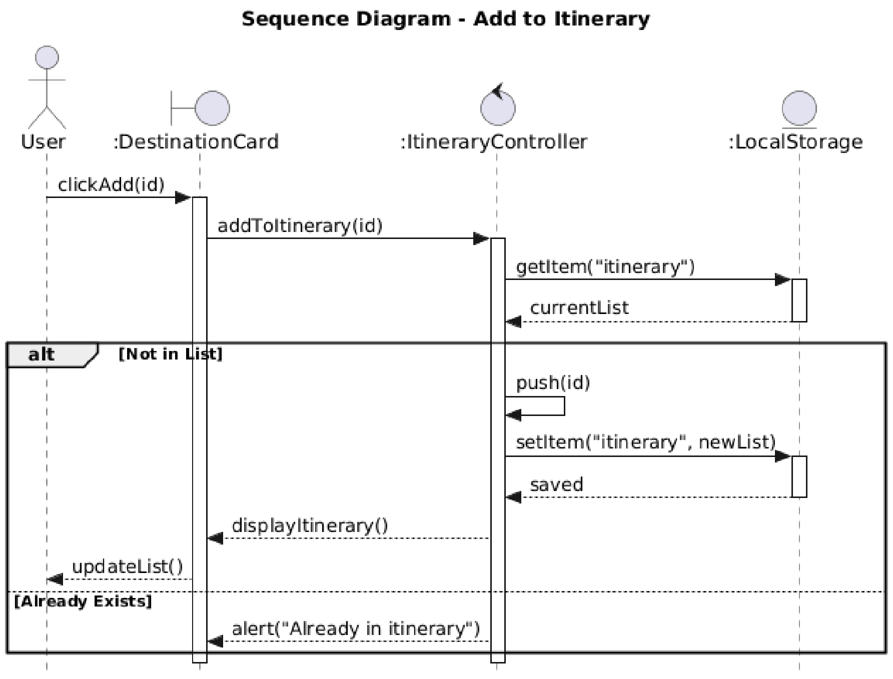


Figure 16: Add Itinerary Sequence Diagram

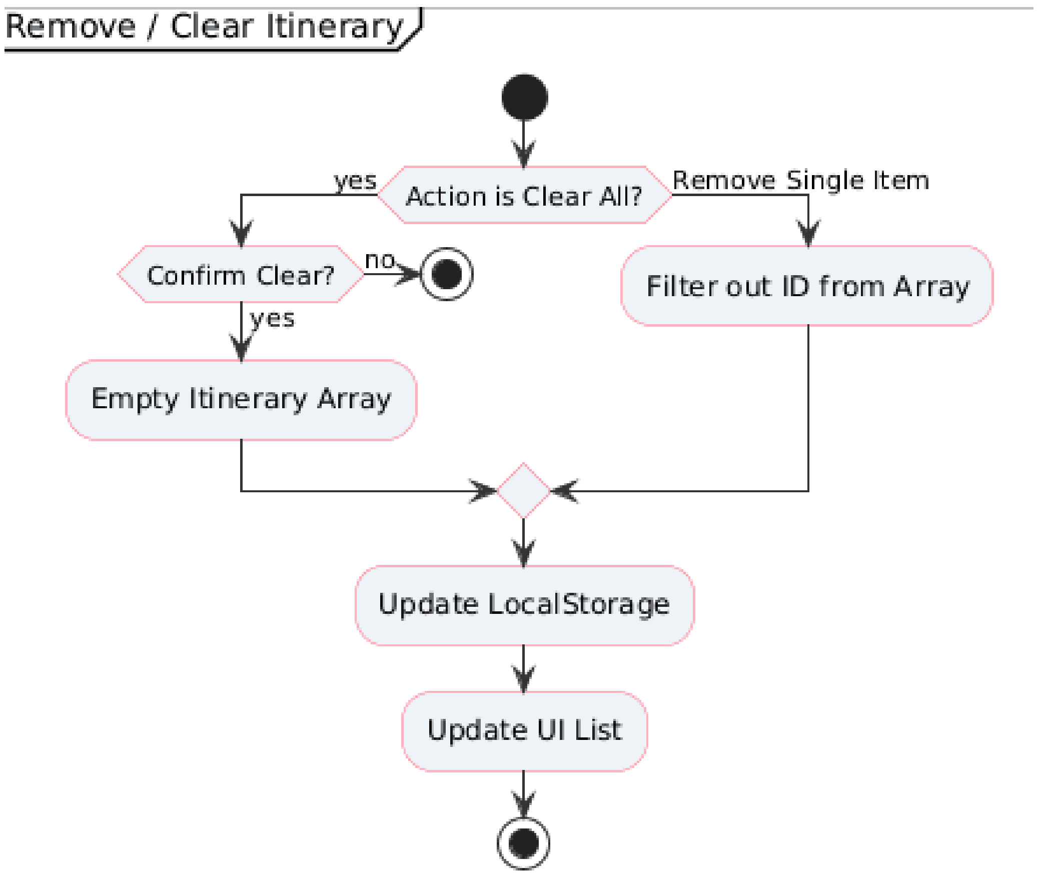


Figure 17: Remove / Clear Itinerary Activity Diagram

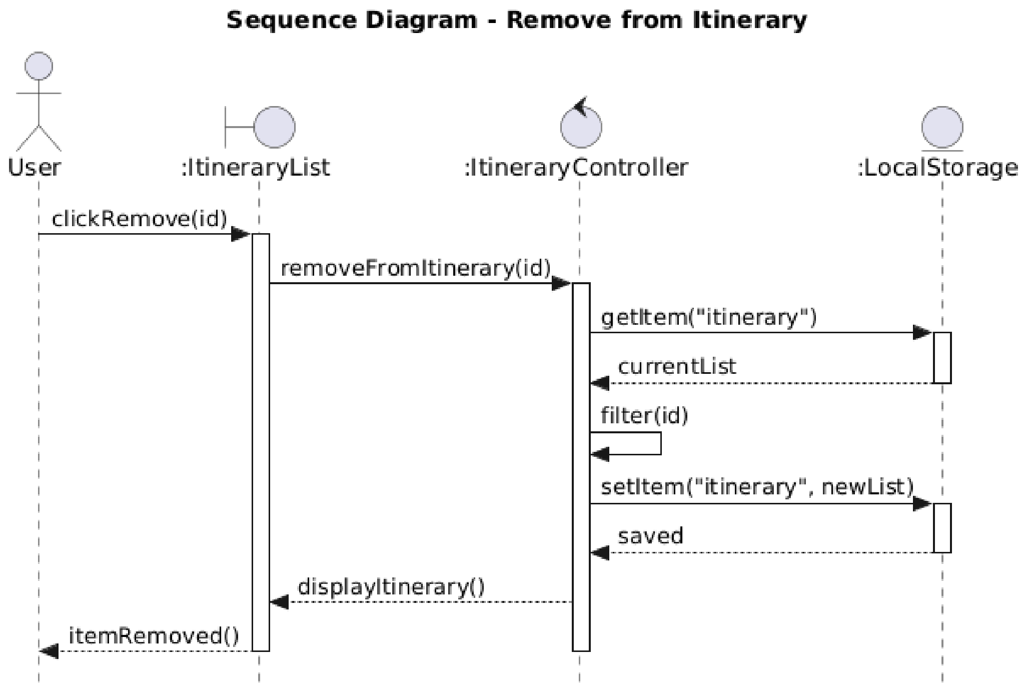


Figure 18: Remove from Itinerary Sequence Diagram

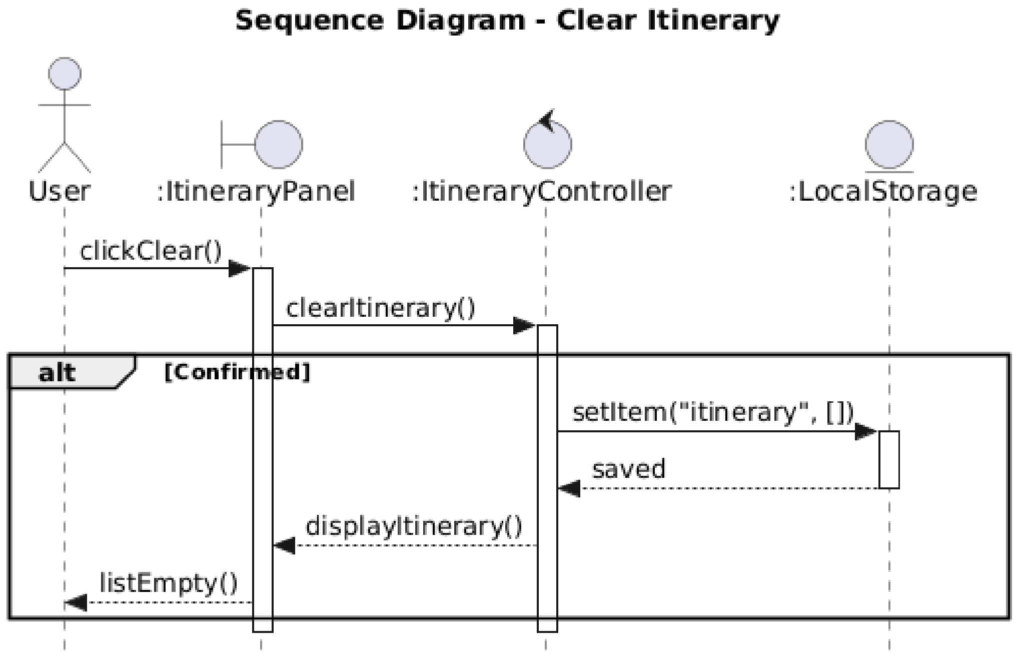


Figure 19: Clear Itinerary Sequence Diagram

3.2.4 Budget Management and Expense Tracking

The Budget Management and Expense Tracking module is based on a dynamic budgeting system that adapts automatically to the user's travel duration at each destination. To achieve this, we defined a daily cost estimate for every destination, calculated using average prices for accommodation (hotel nights), meals, transportation, and other typical expenses. These values represent cost averages, not exact prices, allowing us to provide a realistic yet flexible estimation of what a trip is likely to cost.

All daily cost values are stored in the database. When planning a trip, the user simply enters the number of days they intend to spend at each destination. The system then multiplies the predefined daily cost of the destination by the user's length of stay, generating an immediate and accurate budget estimation. This approach ensures that the total projected budget scales naturally with the user's choices, making the financial planning process both intuitive and tailored.

In addition to the automated estimation, the module also includes an expense tracking feature. Users can manually add extra expenses that may not be included in the base budget—for example souvenirs, shopping, special activities, or unexpected costs. These additional expenses are incorporated into the total budget, giving the user a complete and transparent overview of their actual and estimated spending.

By combining predefined daily costs with user-entered trip durations and optional custom expenses, the Budget Management and Expense Tracking module provides a clear, adaptable, and user-friendly financial planning experience throughout the travel preparation process.

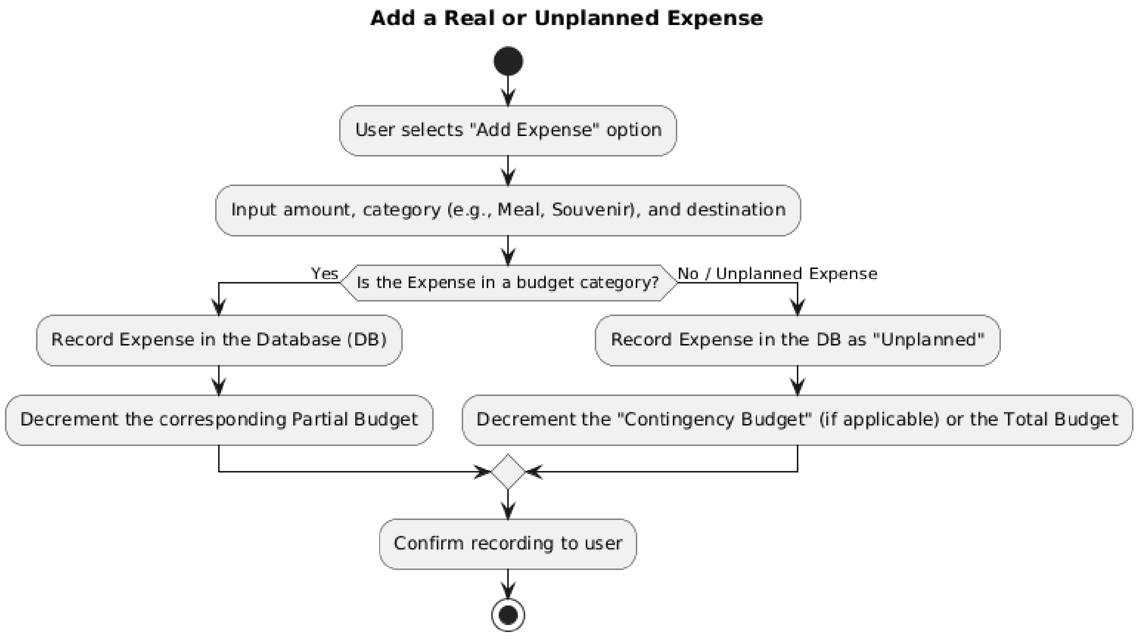

Add a Real or Unplanned Expense
User selects "Add Expense" option
Input amount, category (e.g., Meal, Souvenir), and destination
Is the Expense in a budget category?
Yes
No / Unplanned Expense
Record Expense in the Database (DB)
Record Expense in the DB as "Unplanned"
Decrement the corresponding Partial Budget
Decrement the "Contingency Budget" (if applicable) or the Total Budget
Confirm recording to user

Figure 20: Add Expense Activity Diagram

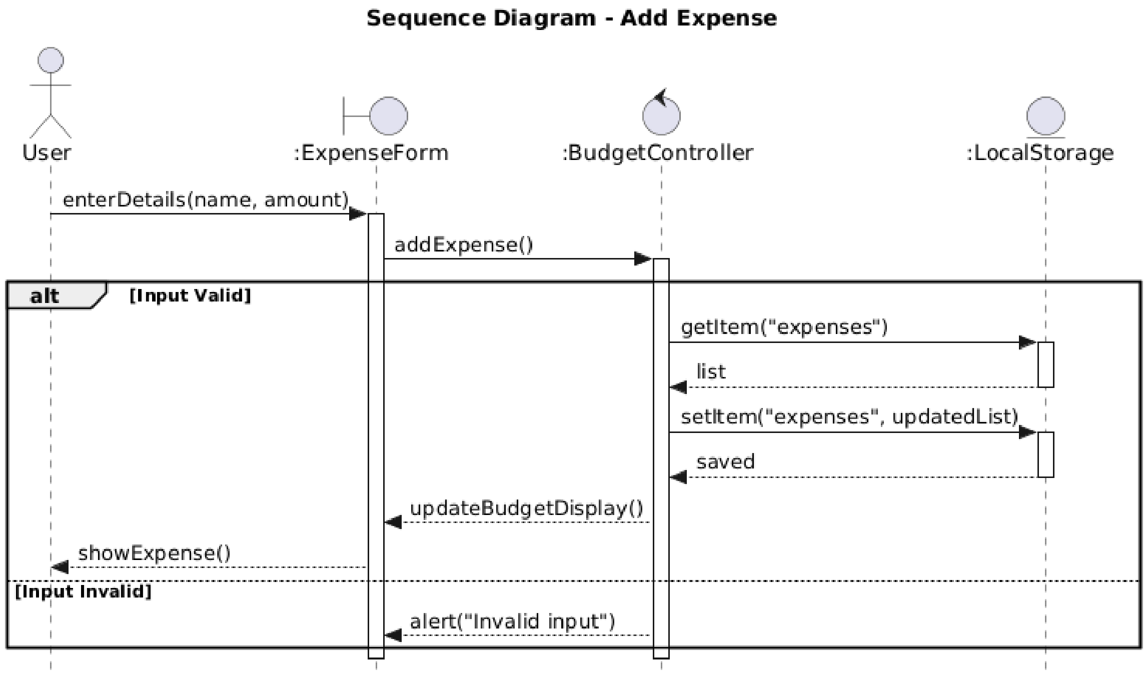


Figure 21: Add Expense Sequence Diagram

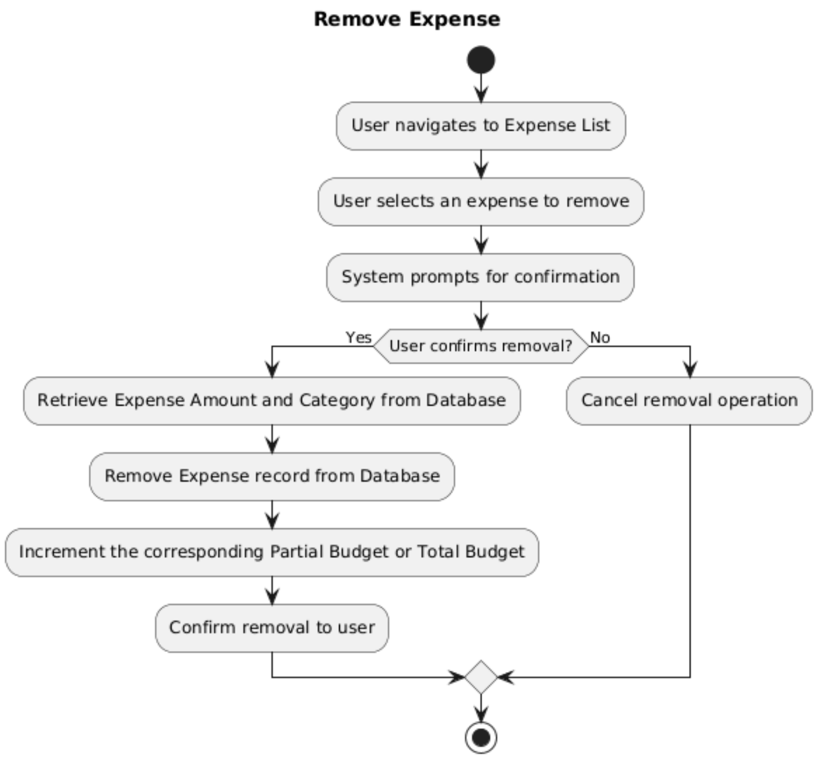

Remove Expense
User navigates to Expense List
User selects an expense to remove
System prompts for confirmation
User confirms removal?
Yes
No
Retrieve Expense Amount and Category from Database
Cancel removal operation
Remove Expense record from Database
Increment the corresponding Partial Budget or Total Budget
Confirm removal to user

Figure 22: Remove Expense Activity Diagram

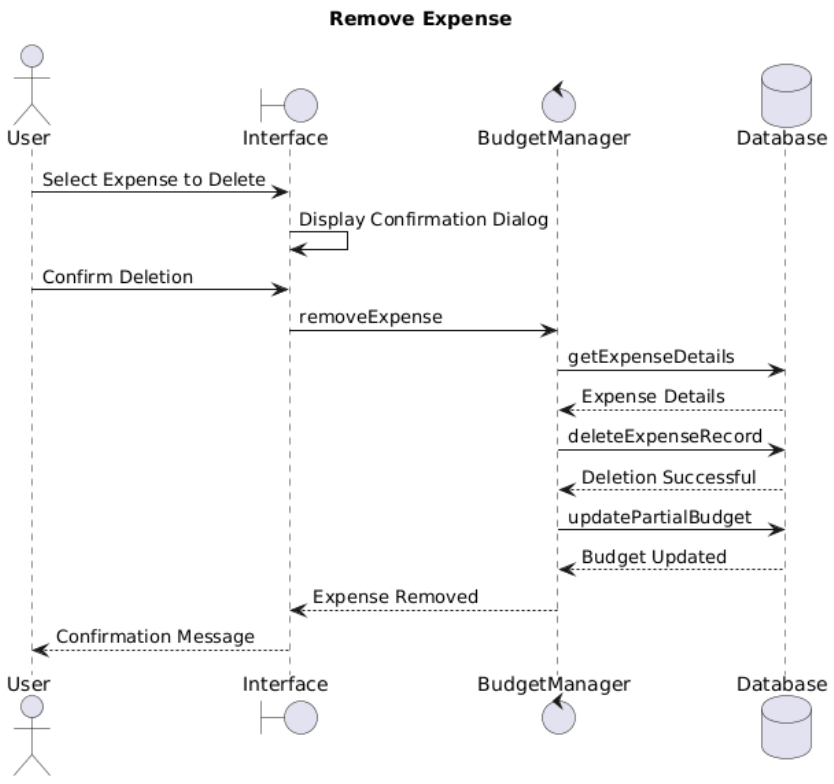


Figure 23: Remove Expense Sequence Diagram

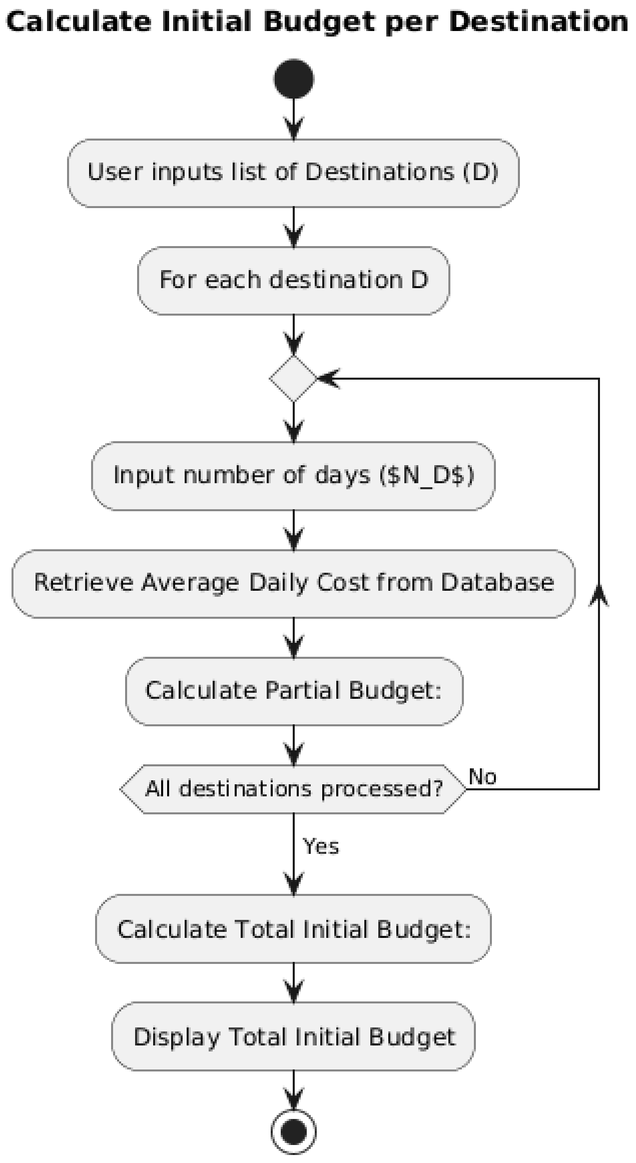

Calculate Initial Budget per Destination
User inputs list of Destinations (D)
For each destination D
Input number of days (N_D)
Retrieve Average Daily Cost from Database
Calculate Partial Budget:
All destinations processed?
No
Yes
Calculate Total Initial Budget:
Display Total Initial Budget

Figure 24: Set Total Budget Activity Diagram

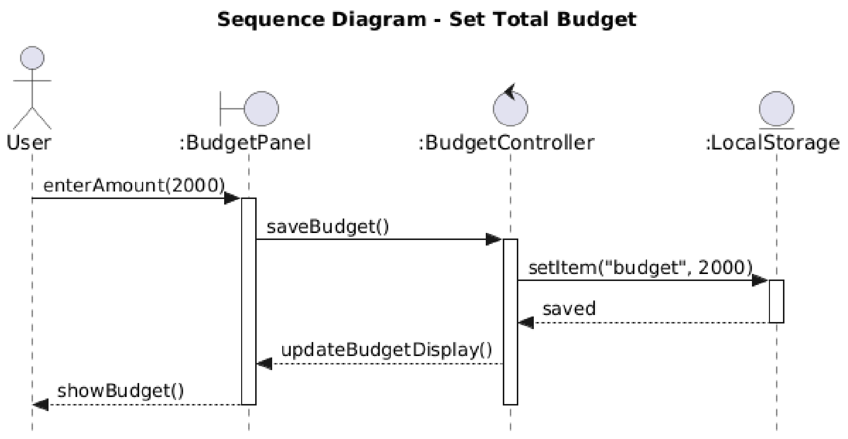


Figure 25: Set Total Budget Sequence Diagram

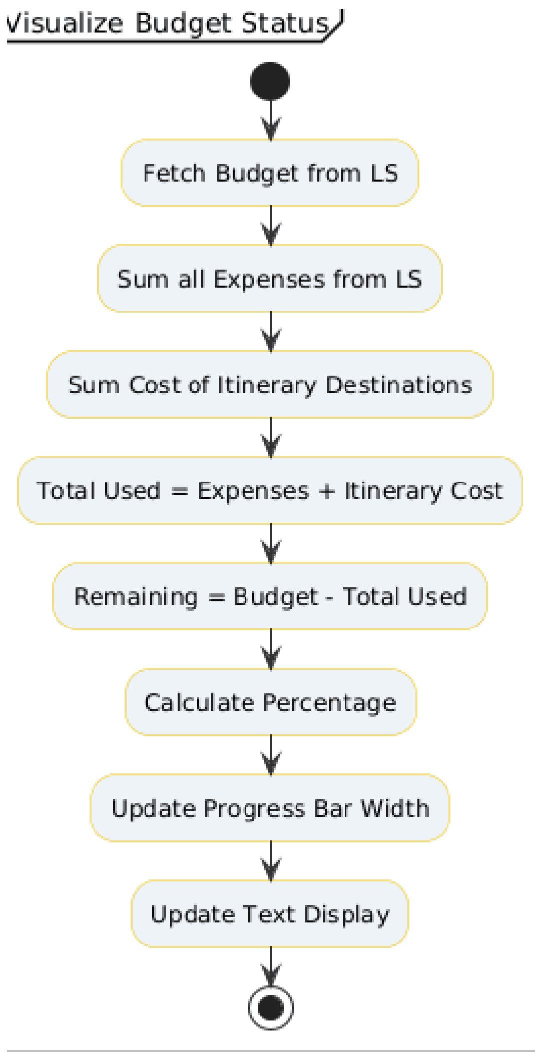

Visualize Budget Status
Fetch Budget from LS
Sum all Expenses from LS
Sum Cost of Itinerary Destinations
Total Used = Expenses + Itinerary Cost
Remaining = Budget - Total Used
Calculate Percentage
Update Progress Bar Width
Update Text Display

Figure 26: Visualize Budget Status Activity Diagram

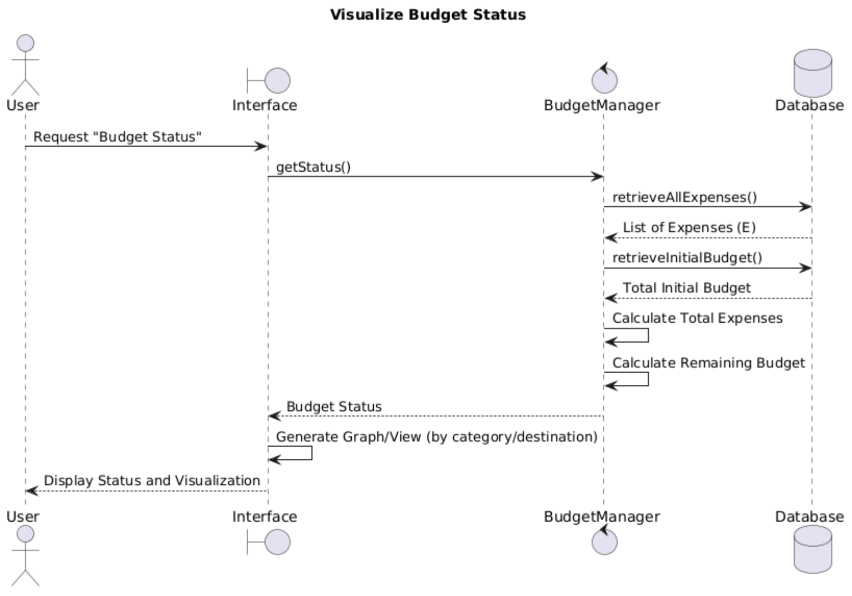


Figure 27: Visualize Budget Status Sequence Diagram

3.2.5 Data Persistence (History and Storage)

FR-HIS-01: Save Trip Configuration

The system shall allow the user to archive their current active itinerary into persistent storage.

- **Trigger:** The user clicks the "Save to Travel History" button located in the itinerary panel.
- **Pre-condition Validation:** The system must verify that the `itinerary` array is not empty. If the array length is 0, the system shall abort the operation and display an alert: "You need at least one destination in your itinerary before saving".
- **User Input:** The system shall prompt the user via a modal dialog to enter a unique name for the trip (default value: "My Trip"). If the user cancels the prompt, the save operation is aborted.
- **Data Processing:**
  - The system shall generate a unique ID based on the current timestamp (`Date.now()`).
  - The system shall calculate the `totalCost` by summing the cost of all destinations currently in the itinerary.
  - The system shall create a "snapshot" of the destinations (including name, country, and cost) to ensure history remains accurate even if the main database changes.
- **Output:** The complete trip object is pushed to the `travelHistory` array in LocalStorage.

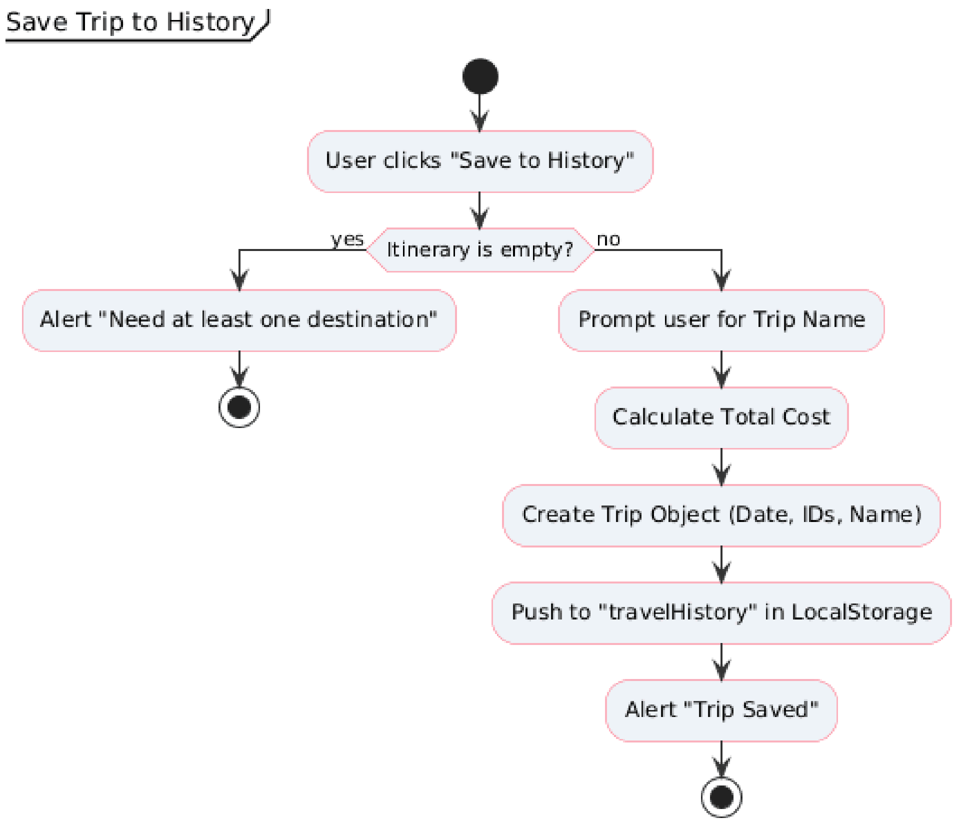

Save Trip to History
User clicks "Save to History"
Itinerary is empty?
yes
no
Alert "Need at least one destination"
Prompt user for Trip Name
Calculate Total Cost
Create Trip Object (Date, IDs, Name)
Push to "travelHistory" in LocalStorage
Alert "Trip Saved"

Figure 28: Save Trip to History Activity Diagram

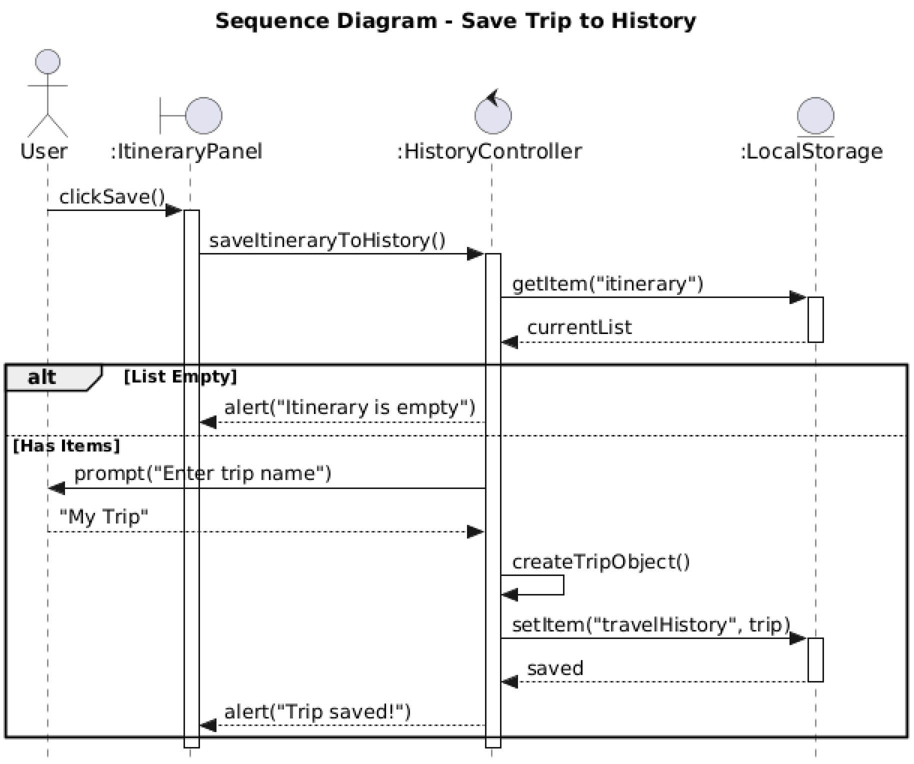


Figure 29: Save Trip to History Sequence Diagram

FR-HIS-02: Render History Log

The system shall display the chronological list of saved trips on the dedicated History page.

- **Trigger:** The user navigates to `history.html` or the page initializes.
- **Sorting Logic:** The system shall display trips in **reverse chronological order** (newest trip at the top) by applying a reverse operation on the history array.
- **Empty State:** If the `travelHistory` array is null or empty, the system shall render a specific message: "You have no saved trips yet" to guide the user.
- **Visual Components:** For each trip, the system shall render a card containing:
  - The Trip Name preceded by an icon.
  - The creation date and the total calculated cost formatted in dollars.
  - A list of destinations with their respective countries and individual costs.

FR-HIS-03: Restoration (Load Trip)

The system shall allow the user to reinstate a past trip as the current active itinerary.

- **Trigger:** The user clicks the "Load" button on a specific history card.
- **Processing:**
  - The system shall locate the trip in the `travelHistory` array using its unique ID.
  - The system shall extract the destination IDs from the saved trip.
  - The system shall **overwrite** the current `itinerary` key in LocalStorage with these extracted IDs.
- **Feedback:** The system shall alert the user ("Trip [Name] loaded into your itinerary") to confirm the action.

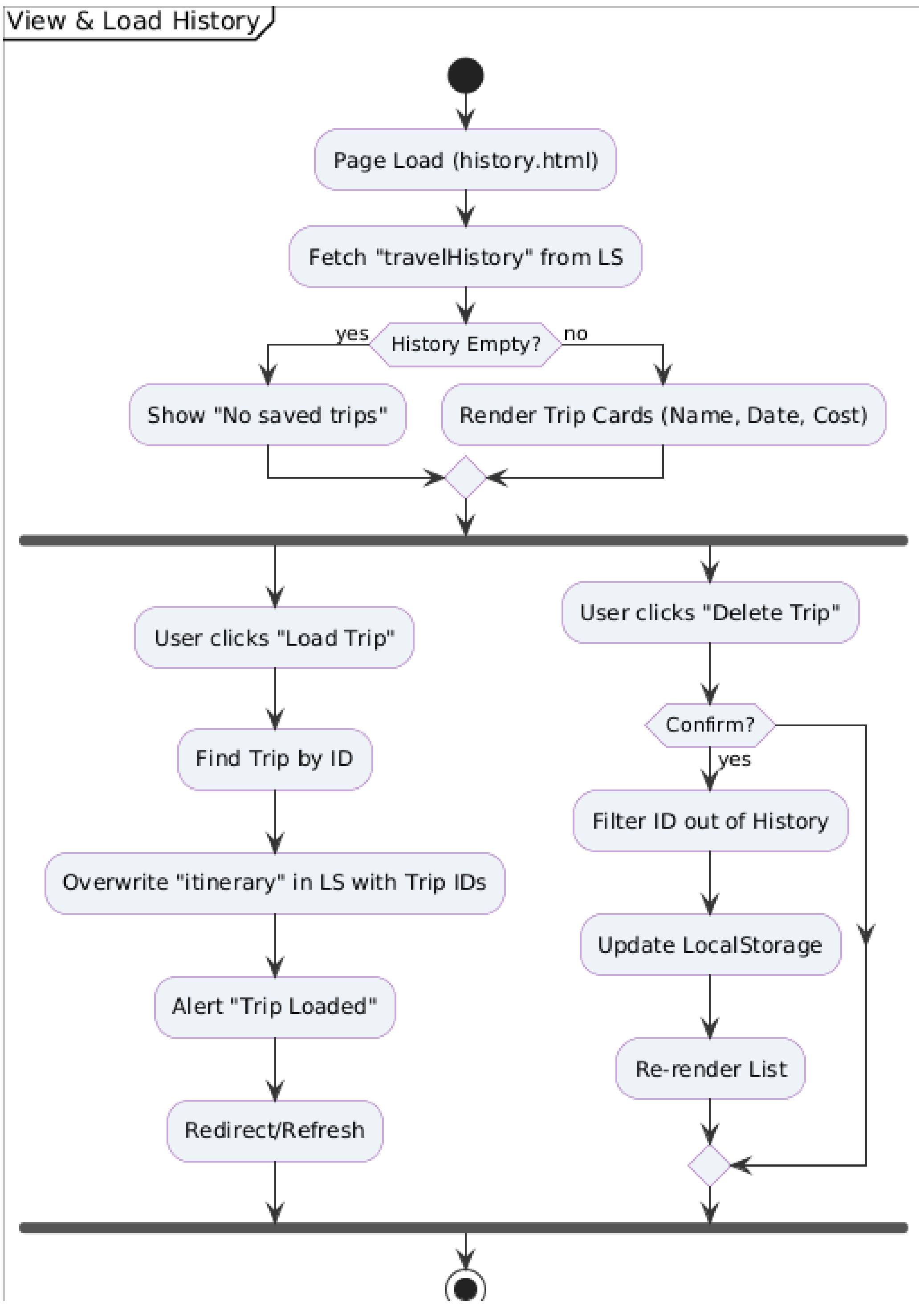


Figure 30: View and Load History Activity Diagram

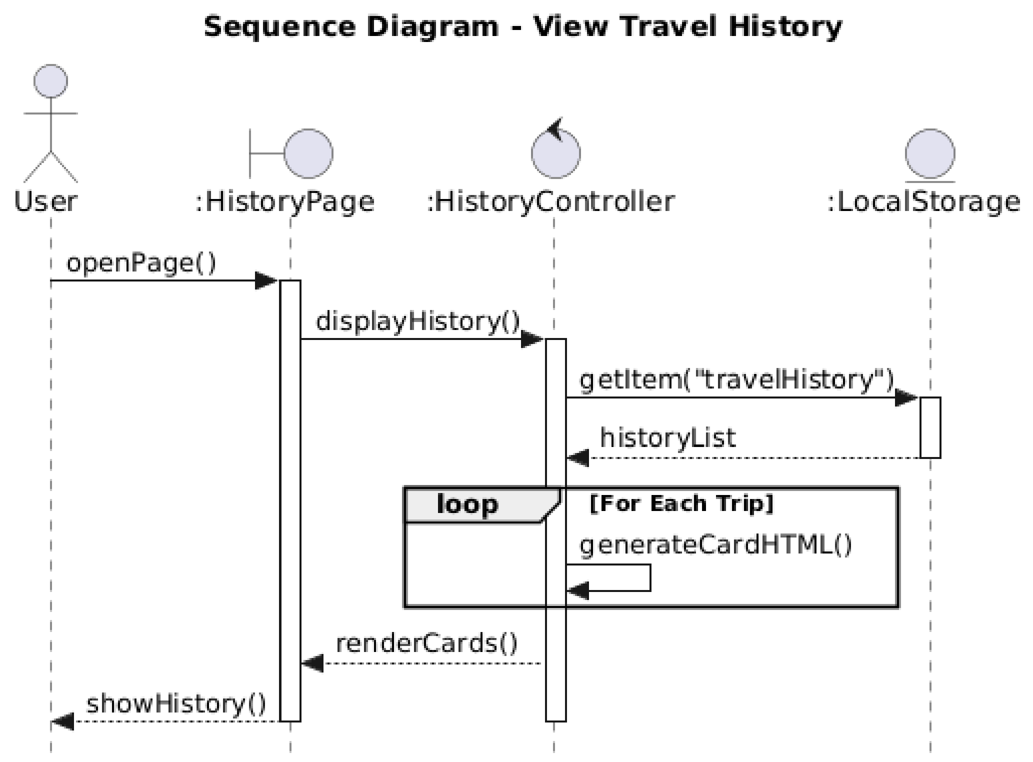


Figure 31: View Travel History Sequence Diagram

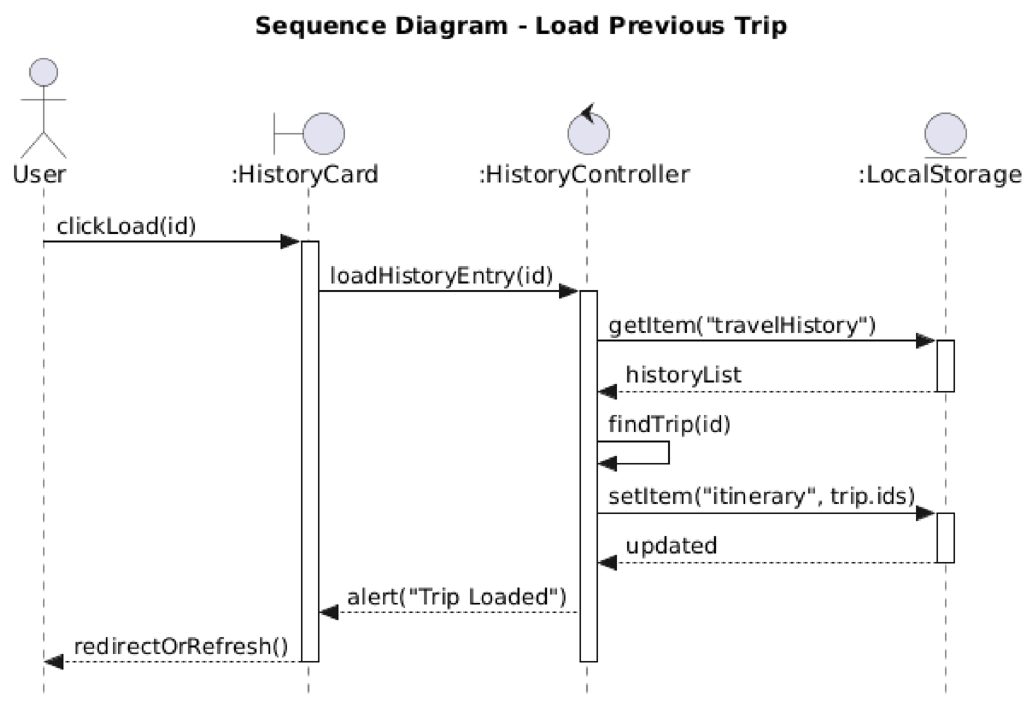


Figure 32: Load Previous Trip Sequence Diagram

FR-HIS-04: Trip Sharing Mechanism

The system shall allow users to share a specific itinerary configuration via a generated link.

- **Trigger:** The user clicks the "Share" button on a history card.
- **URL Generation:**
  - The system shall create a JSON payload containing the trip name, date, and destination IDs.
  - The system shall encode this payload into a **Base64 string** using `btoa()` and `encodeURIComponent()` to ensure URL safety.
  - The system shall append this string as a query parameter (`?sharedTrip=...`) to the application's base URL.
- **Clipboard Action:** The system shall attempt to write the generated URL directly to the user's system clipboard.
- **Fallback:** If the Clipboard API fails, the system shall copy the raw JSON object text instead and notify the user.

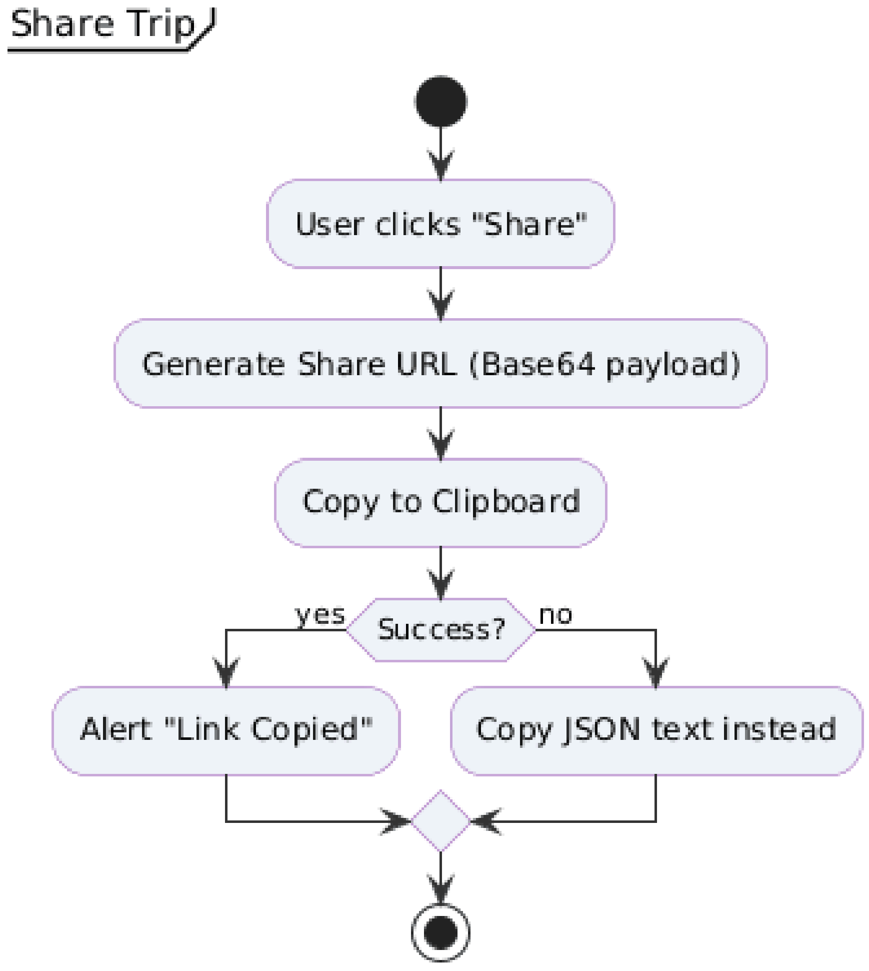


Figure 33: Share Trip Activity Diagram

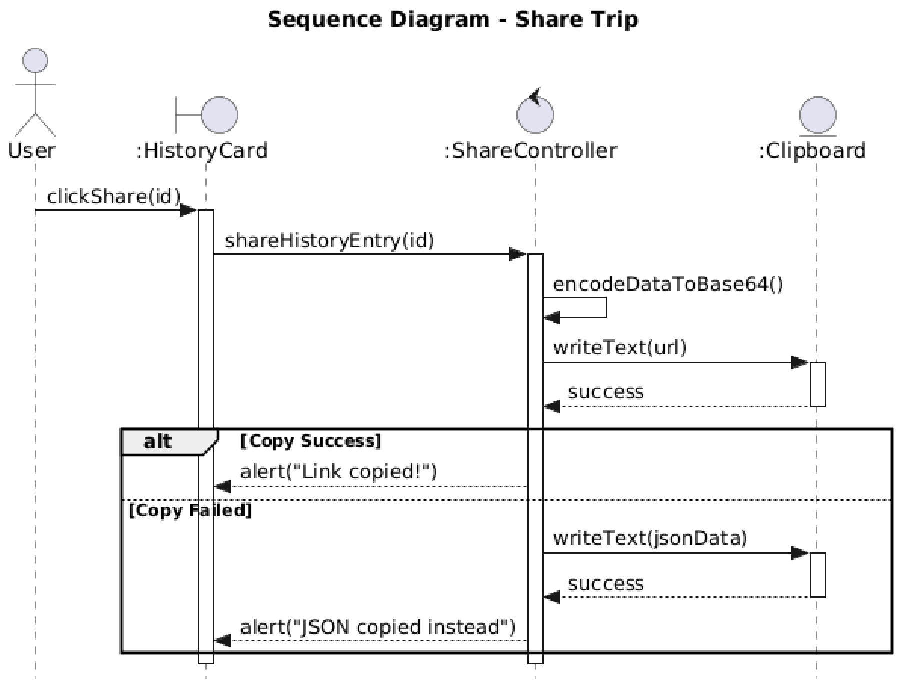


Figure 34: Share Trip Sequence Diagram

FR-HIS-05: Permanent Deletion

The system shall allow users to remove obsolete entries from their history.

- **Trigger:** The user clicks the "Delete" button on a history card.
- **Safety Mechanism:** The system must display a confirmation dialog ("Delete this saved trip?") before processing.
- **Data Update:** Upon confirmation, the system shall filter the `travelHistory` array to exclude the target ID and update LocalStorage immediately.
- **UI Update:** The system shall re-render the history list immediately to reflect the removal without requiring a page reload.

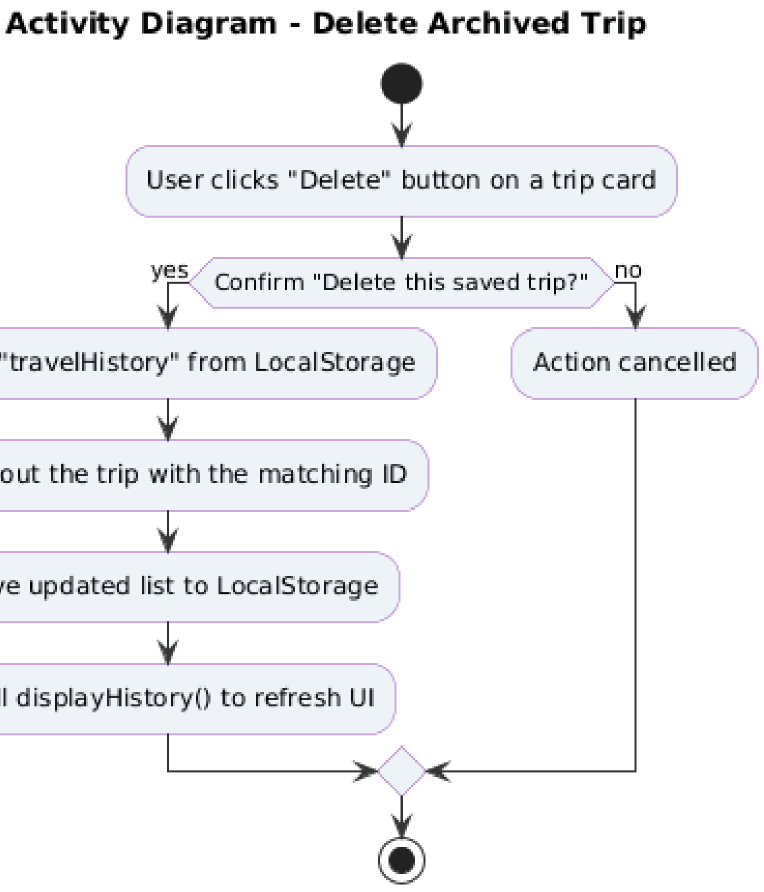

Activity Diagram - Delete Archived Trip
User clicks "Delete" button on a trip card
Confirm "Delete this saved trip?"
yes
no
Fetch "travelHistory" from LocalStorage
Action cancelled
Filter out the trip with the matching ID
Save updated list to LocalStorage
Call displayHistory() to refresh UI

Figure 35: Delete Archived Trip Activity Diagram

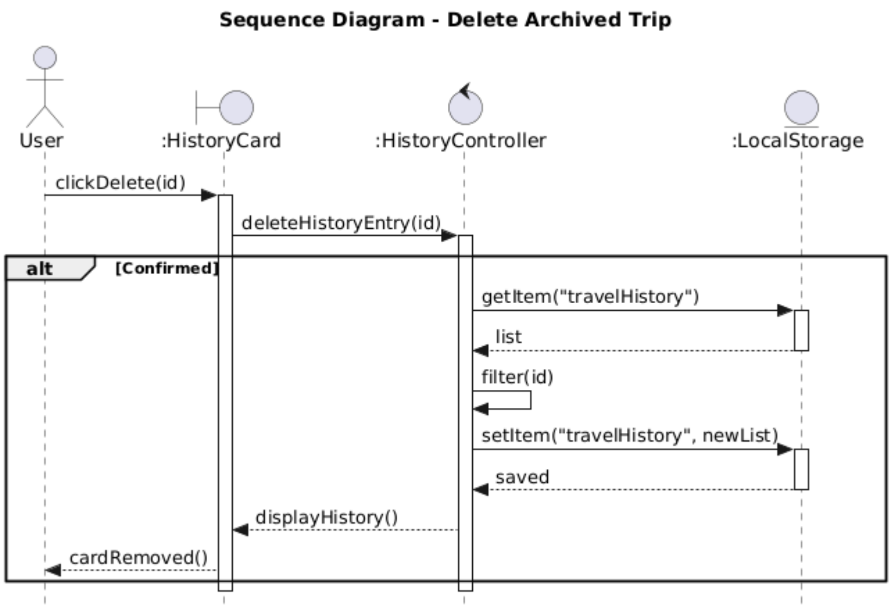


Figure 36: Delete Archived Trip Sequence Diagram

3.2.6 Favorites

FR-FAV-01: Add to Favorites

- **Trigger:** The user clicks the “Favorite” button on a destination card.
- **Alert Mechanism:** The system display a validation dialog *” Destination has been added to your favorites !”* .
- **Data Update:** The ID of the destination shall be appended to the favorites array stored in the browser’s `localStorage`. If the destination ID is already present in the favorites array, no action shall be taken on the data.
- **UI Update:** The system shall re-render the favorite list immediately to reflect the addition without requiring a page reload.

FR-FAV-02: Display Favorites

- **Trigger :** The page finishes loading and initializes. An action modifies the favorites data (e.g, adding or removing a favorite).
- **Logic :** The system shall retrieve the list of destination IDs from the favorites key in `localStorage`. The system shall iterate through the list of IDs and retrieve the full destination object (name, country, image) from the global destinations array
- **Data Validation :** If the favorites array is empty, the system shall display a ”*No favorites yet*” message. If a stored ID does not correspond to a valid destination in the destinations array, the system shall ignore that ID.

FR-FAV-03: Remove Favorites

- **Trigger :** The user clicks the ”Remove” button next to a destination in the favorites list.
- **Alert Mechanism :** The system shall delete the favorite immediately without requiring a confirmation dialog.
- **Data Update :** The ID of the destination shall be removed from the favorites array stored in the browser’s `localStorage`. The updated favorites array shall overwrite the previous version in `localStorage`.
- **UI Update :** The system shall re-render the favorite list immediately (`displayFavorites`) to reflect the removal.

FR-FAV-04 : Clear All Favorites

- **Trigger :** The user clicks the ” Clear All ” button in the Favorites section.
- **Alert Mechanism :**

- If the favorites list is already empty, the system shall alert, "Your favorites list is already empty."
- If the list is populated, the system shall prompt the user with a confirmation dialog before proceeding with deletion.
- Upon successful completion, the system may provide a success alert (eg, "All favorites have been cleared.").

- **Data Update :** If confirmed, the system shall reset the favorites key in `localStorage` to an empty array.
- **UI Update :** The system shall re-render the favorite list immediately to reflect the reset without requiring a page reload.

FR-FAV-05 : Share Favorites

- **Trigger :** The user clicks the " Share List " button in the Favorites section.
- **Logic and data collection :** If the favorites array is empty, the system shall alert, "Nothing to export!" and stop. Otherwise, the system shall iterate through the favorites IDs, retrieve destination details (name, country, cost) from the destinations array, and format the data into a plain text string.
- **Data Action :** The system shall attempt to copy the formatted text string to the user's clipboard using the `navigator.clipboard.writeText` API.
- **Alert Mechanism :** Upon successful copy, the system shall alert, "Favorites list copied to clipboard!". If the copy fails (due to permissions or browser restrictions), the system shall inform the user of the failure.

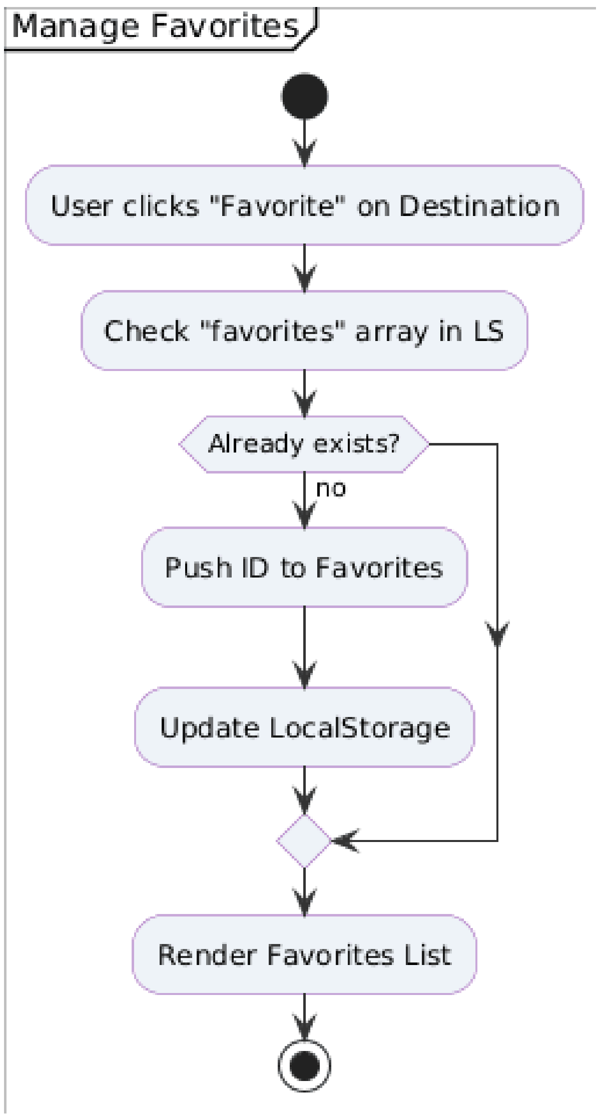


Figure 37: Manage Favorites Activity Diagram

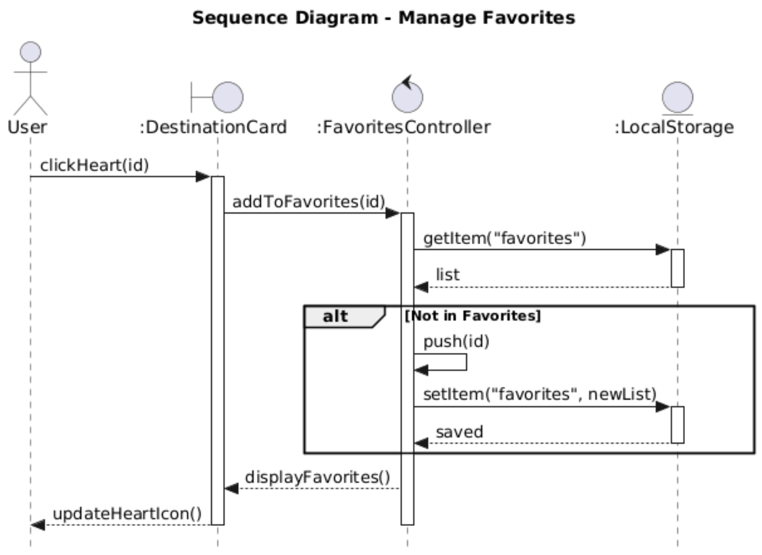


Figure 38: Manages Favorites Sequence Diagram

3.2.7 Reviews

FR-REV-01: Submit a Review

- **Trigger:** The user selects a star rating and clicks the "Submit" button after entering a comment.
- **Data Validation:** The system shall verify that a rating has been selected and that the comment field is not empty. If one of these conditions is not met, an alert message shall be displayed: *"Please add both a rating and a comment!"*.
- **Data Collection:** The system shall retrieve the current user's name and email from the `localStorage` user object. If no user is found, the review shall be associated with a *Guest* user.
- **Data Update:** A new review object (ID, username, email, rating, comment, creation date) shall be appended to the reviews array stored in the browser's `localStorage`.
- **UI Update:** The system shall immediately re-render the reviews list and reset the review form (comment input and star selection) without requiring a page reload.

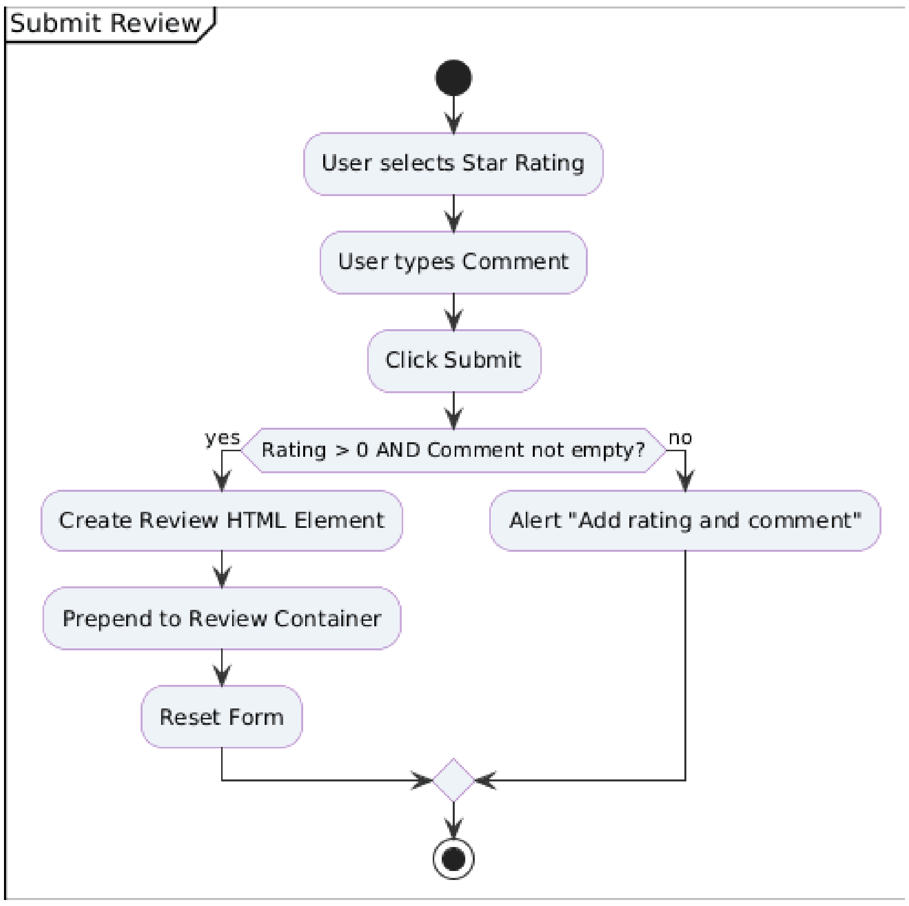

Submit Review
User selects Star Rating
User types Comment
Click Submit
Rating > 0 AND Comment not empty?
yes
no
Create Review HTML Element
Alert "Add rating and comment"
Prepend to Review Container
Reset Form

Figure 39: Submit Review Activity Diagram

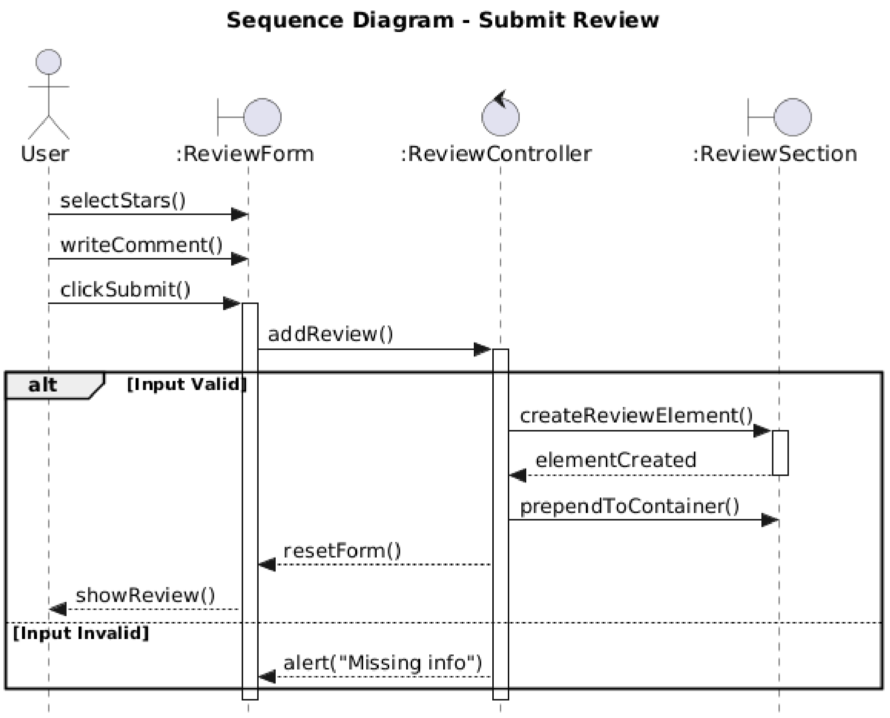


Figure 40: Submit Review Sequence Diagram

FR-REV-02: Load and Display Reviews

- **Trigger:** The page finishes loading or the reviews data is updated (add, delete, filter, or sort).
- **Logic:** The system shall retrieve the reviews array from the `localStorage` key `waygo_reviews`.
- **Data Validation:** If the stored data cannot be parsed or is invalid, the system shall initialize an empty reviews array.
- **UI Update:**
  - If no reviews are available, the system shall display the message *"No reviews yet. Be the first to share your experience!"*.
  - Otherwise, the system shall render each review card with the username, date, star rating, and comment.

FR-REV-03: Filter Reviews by Rating

- **Trigger:** The user clicks on a star rating row in the rating distribution panel.
- **Logic:** The system shall filter the reviews list to display only reviews matching the selected star rating.
- **Toggle Behavior:** If the user clicks the same rating twice, the system shall remove the filter and display all reviews.

- **UI Update:** The reviews list shall be re-rendered immediately to reflect the active filter.

FR-REV-04: Sort Reviews

- **Trigger:** The user changes the selected option in the reviews sorting dropdown.
- **Sorting Modes:**
  - *Relevance (descending):* Higher ratings first, then newer reviews.
  - *Relevance (ascending):* Lower ratings first, then older reviews.
  - *Date (descending):* Newest reviews first.
  - *Date (ascending):* Oldest reviews first.
- **Logic:** The system shall apply the selected sorting algorithm to the filtered reviews list.
- **UI Update:** The sorted reviews shall be displayed immediately without reloading the page.

FR-REV-05: Delete a Review

- **Trigger:** The user clicks the "Delete" button on one of their own reviews.
- **Access Control:** The delete option shall only be visible for reviews created by the currently logged-in user (email matching).
- **Data Update:** The selected review shall be removed from the reviews array stored in `localStorage`.
- **UI Update:** The system shall re-render the reviews list and update the review summary immediately.

FR-REV-06: Display Review Summary and Statistics

- **Trigger:** Reviews are loaded, added, deleted, filtered, or sorted.
- **Logic:** The system shall compute:
  - The total number of reviews.
  - The average rating score.
  - The percentage distribution for each star rating (1 to 5).
- **UI Update:** The system shall display the average score, visual star representation, total review count, and interactive rating bars.
- **Interaction:** Clicking on a rating bar shall activate the corresponding star filter.

3.3 Non-Functional Requirements

3.3.1 Performance

These requirements ensure the system is responsive and efficient under various user loads.

- **NFR-PERF-01 / Search Response Time :**

  The response time for the `Search Destination` function must be less than 500 milliseconds when searching a database containing up to 1000 destinations.

- **NFR-PERF-02 / Map Loading Time :**

  The initialization of the Leaflet map and the display of markers for all destinations `initMap` must be completed in under 3 seconds.

- **NFR-PERF-03 / Budget Processing Speed :**

  Adding or removing an expense `addExpense`, `removeExpense` and updating the budget progress bar `updateBudgetDisplay`must be instantaneous, causing no noticeable delay (under 100 milliseconds).

- **NFR-PERF-04 / Display Scalability :** The display of the travel history `displayHistory` must remain fluid even with 50 saved trips.

3.3.2 Privacy

These requirements focus on the protection of user data, even though storage is local `localStorage`.

- **NFR-PRIV-01 / Data Separation :** User personal information (Name, Email) must be stored separately from the active login session data in `localStorage` to facilitate logging out without deleting the permanent account.
- **NFR-PRIV-02 / Sensitive Data :** No sensitive banking or payment information shall be collected or stored. The budget manager only handles raw numerical amounts.
- **NFR-PRIV-03 / Guest Anonymity :** Unconnected users (Guests) shall not be able to access restricted features such as budget management, travel history, or favorites management `checkLoginStatus`.
- **NFR-PRIV-04 / Comment Protection :** Only the user who posted a review (identified by `r.userEmail` must have the permission to delete it `deleteReview`.

3.3.3 Reliability

- **NFR-REL-01 / Media Error Handling :** If a destination image (d.image) fails to load, the system must display a default state (e.g., reduced opacity and an icon) instead of breaking the destination card (onerror).
- **NFR-REL-02 / Input Validation :** All numerical inputs (e.g., budget or expenses) must be properly validated to ensure they are positive, non-zero numbers before being saved.
- **NFR-REL-03 / State Persistence :** Critical data (Itinerary, Favorites, Budget, Expenses, History, and Reviews) must be stored in `localStorage` to persist across browser sessions.
- **NFR-REL-04 / Missing Data Management :** The system must gracefully handle cases where data is missing, for example, by assigning a cost of 0€ if a destination lacks a cost field during budget or itinerary calculation.

4 System Design

This section describes the overall structure of the Waygo application and how its components interact. Given that Waygo is a client-side application using the local browser storage, the architecture is relatively simple.

4.1 System Architecture

Waygo uses a two-tier client-side architecture :

- **Frontend :** Everything runs entirely in the user's browser, handling the UI, presentation logic and user imputs. This includes the HTML, CSS and the majority of the JavaScript functions.
- **Data Persistence :** This part is responsible for storing and retrieving all the application state, activate as a "database". This layer is composed of local data and browser storage. While local data only represent the `data.js` file storing a fixed list of destinations and their details, the browser storage is used to store user-specific, changing data such as favorites, itinerary, expenses, history and user session details.

The application is decomposed into several independent JavaScript modules, each responsible for a specific feature and communicating mainly through the global variable `destinations` and `localStorage`.

The Waygo application has **two core modules** : `data.js` that defines th fixed dataset and `escape and init.js` that manages the application startup and order. The other files are **feature modules** specific to a certain functionality.

4.2 User Interface (UI) Design

This section focuses on the visual presentation and user interaction elements.

The design of Waygo uses a clean, modern aesthetic with a consistent header and primary action buttons styled by `styles.css`. Navigation is managed via the static header bar.

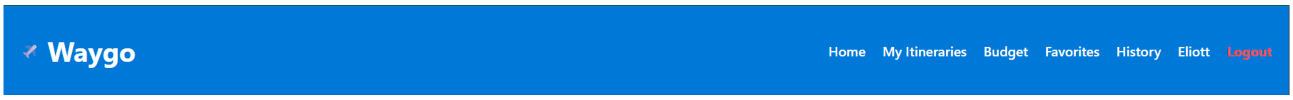
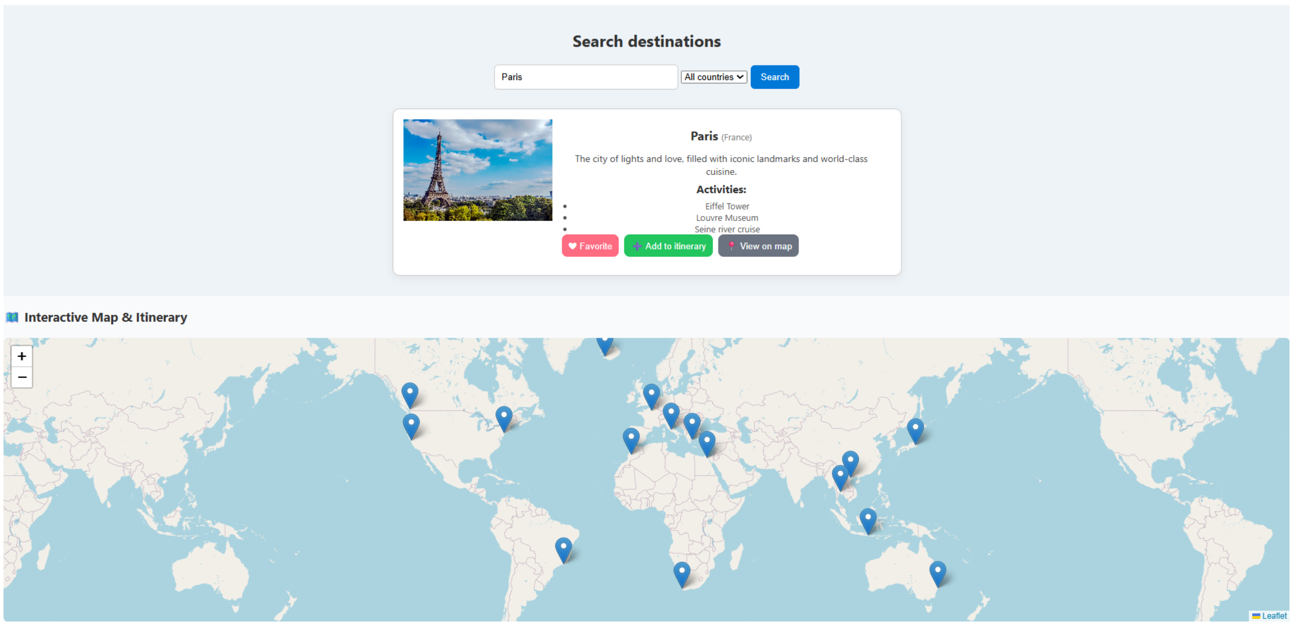


Figure 41: Header and Navigation

4.2.1 Key features interface

The search area is prominent at the top of the page, featuring an input box and a country filter dropdown. The destination cards clearly display the image, name, country, description and recommended list of activities. Key actions (Favorites, Add to Itinerary, View on Map) are directly available on each card.



Paris (France)

The city of lights and love, filled with iconic landmarks and world-class cuisine.

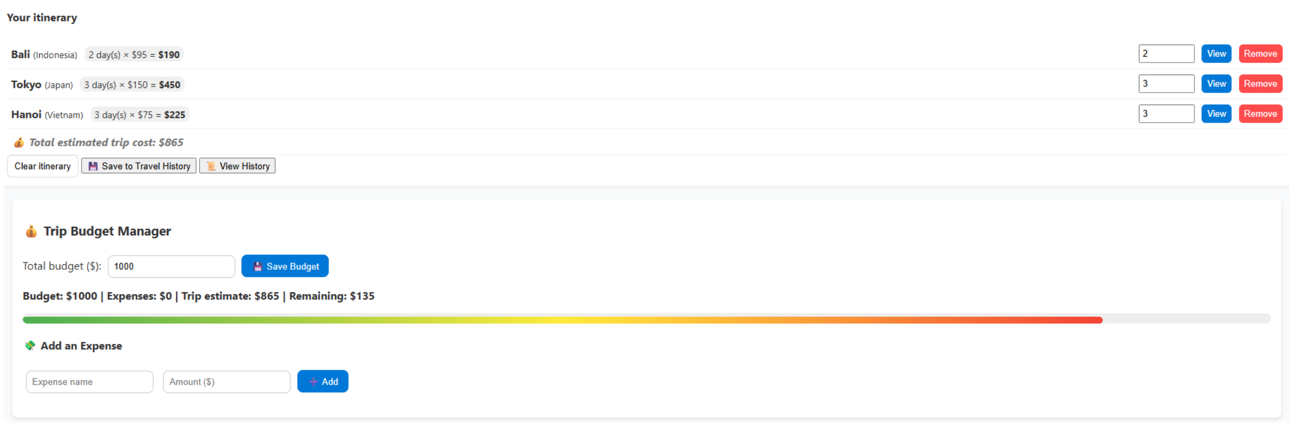
Activities:

- Eiffel Tower
- Louvre Museum
- Seine river cruise

Figure 42: Destination search and Dynamic Map

The map is displayed using Leaflet. The route list is placed directly below the map, showing planned stops, estimated cost and quick removal/view actions.

The budget section is clearly divided into the main progress tracker (Total Budget vs. Used) and the expense entry form, ensuring users can quickly grasp their financial status.



Your itinerary

Bali (Indonesia)	2 day(s) × $95 = **$190**	2
Tokyo (Japan)	3 day(s) × $150 = **$450**	3
Hanoi (Vietnam)	3 day(s) × $75 = **$225**	3

Total estimated trip cost: $865

Trip Budget Manager

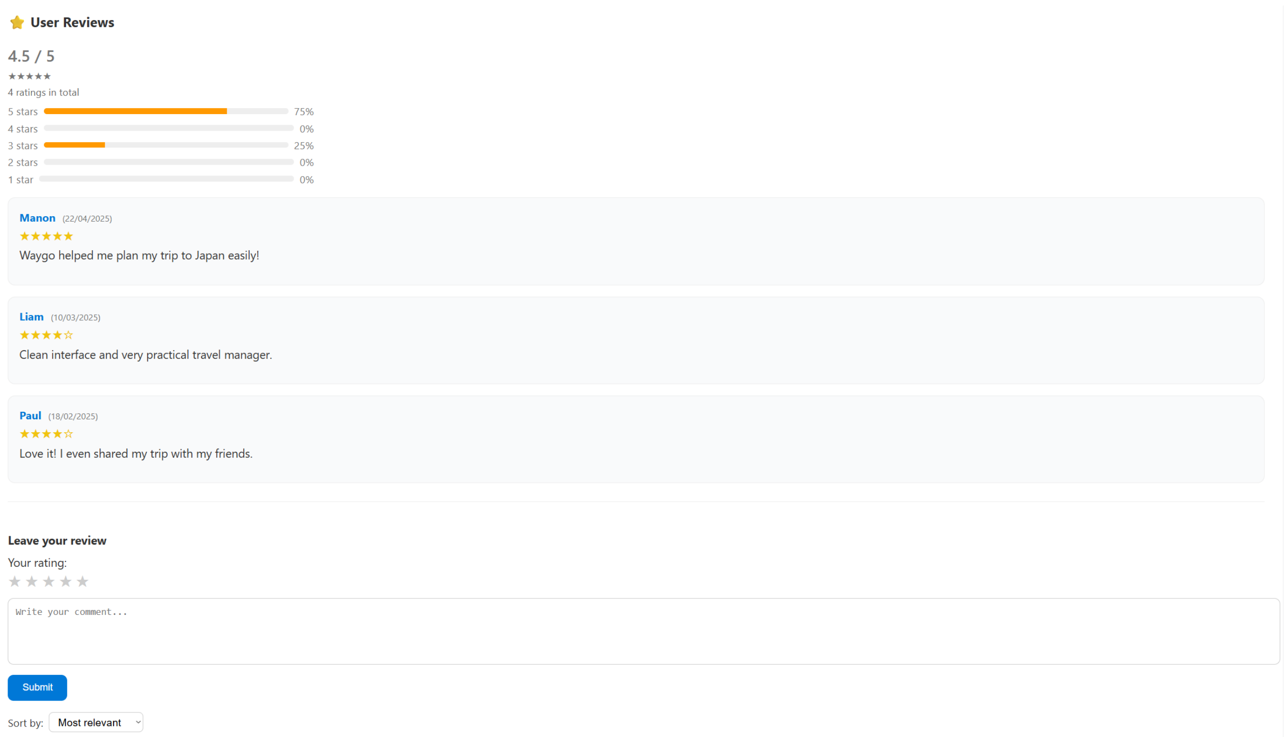
Total budget ($): 1000

Budget: $1000 | Expenses: $0 | Trip estimate: $865 | Remaining: $135

Add an Expense

Figure 43: Itinerary and Budget Management

The review section features a summary, an input box and a rating star system.



User Reviews

4.5 / 5

★★★★★

4 ratings in total

5 stars	75%
4 stars	0%
3 stars	25%
2 stars	0%
1 star	0%

Manon (22/04/2025)
★★★★★
Waygo helped me plan my trip to Japan easily!

Liam (10/03/2025)
★★★★☆
Clean interface and very practical travel manager.

Paul (18/02/2025)
★★★★☆
Love it! I even shared my trip with my friends.

Leave your review

Your rating:
★ ★ ★ ★ ★

Write your comment...

Submit

Sort by: Most relevant

Figure 44: Waygo reviews section

4.2.2 Auxiliary Pages

- Login / Authentication : The login page uses the `auth-card` and `auth-container` classes for a centered, clean look, consistent with the main site's style.

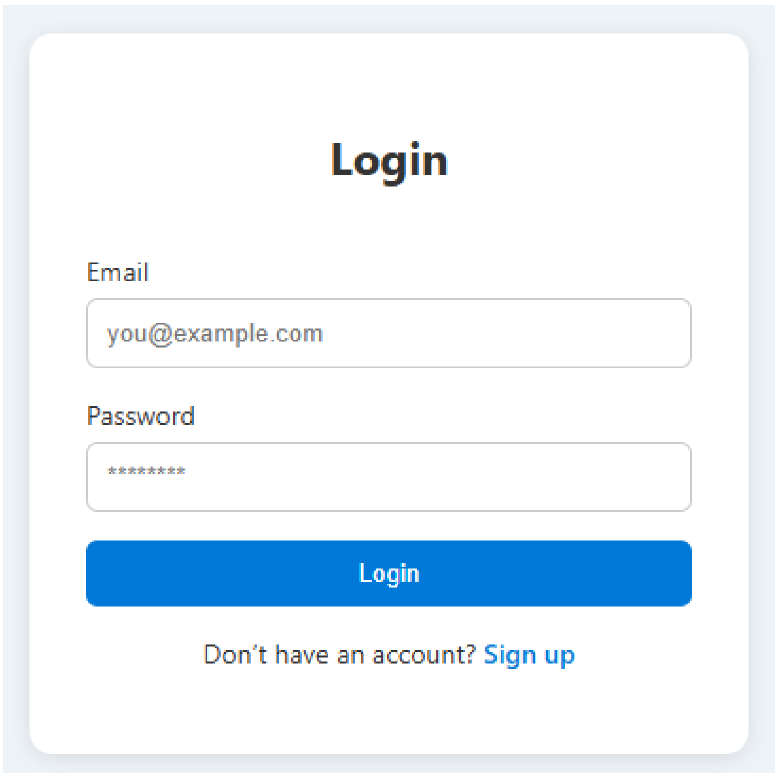
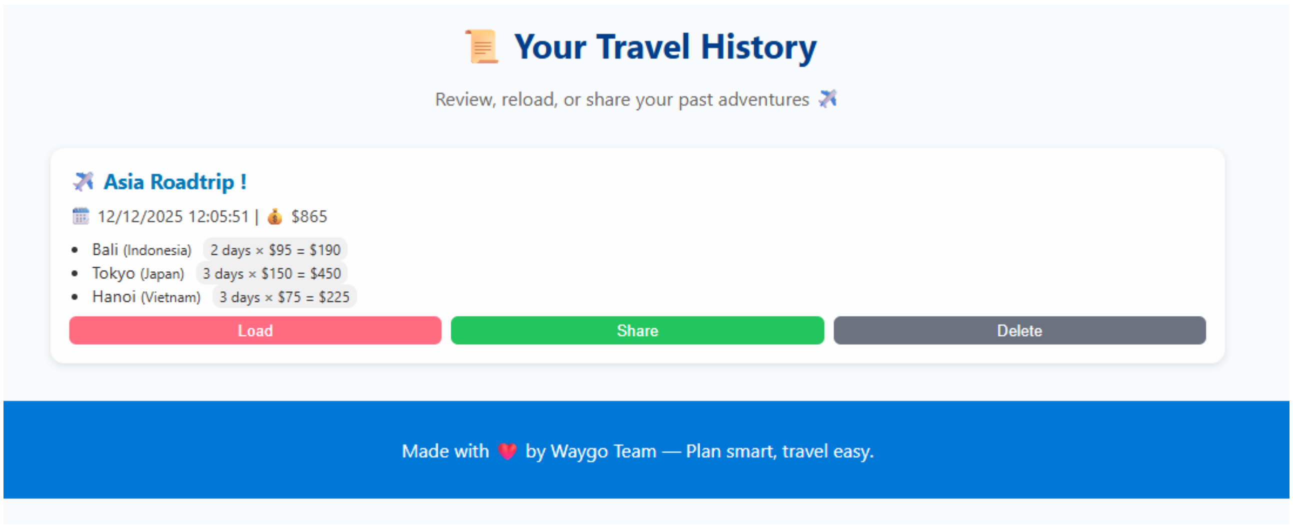


Figure 45: Authentication Page

- History : The dedicated history page displays saved trips in an organized, card-based layout, allowing users to Load, Share, or Delete previous itineraries.



Asia Roadtrip !

12/12/2025 12:05:51 | $865

- Bali (Indonesia) 2 days × $95 = $190
- Tokyo (Japan) 3 days × $150 = $450
- Hanoi (Vietnam) 3 days × $75 = $225

Made with ❤ by Waygo Team — Plan smart, travel easy.

Figure 46: History Page

5 Project Management

This section will summarize the methodologies, planning, and collaboration tools used during the creation of our Waygo project.

5.1 Project Roadmap

This project was executed over approximately two months. The team was formed in early October and the project selection was finalized around October 15th. As the deadline was set for December 15, we had two months to complete this project. Here are the four phases that we followed :

- Phase 1 : Analysis

The project began with the analysis phase. We identified the problem that we wanted to solve and how we could do it. However, since the team had very limited prior experience with version control and web development, the first week was dedicated to defining a realistic scope for the project that was more aligned with our technical capabilities. We focused on setting up our development environments, learning the basics of Github for collaboration, and drafting the initial SRS. This project phase was necessary to the team, prior to writing any code.

- Phase 2 : Interface Design and Prototyping

Once the objectives were clear for everyone, we transitioned to the design phase. It felt easier and reassuring to begin with the visual structure. We used the HTML and CSS language to build this visual layer with no functionality at first. This step allowed us to get more comfortable with coding without being immediately lost in more complex problems later encountered with the logic of JavaScript functionality implementation.

- Phase 3 : Implementation and Integration

The third phase was the longest and the most difficult. We focused on implementing the functionality to the application frontend. A major milestone during this period was the successful implementation of the Leaflet.js map and the implementation of LocalStorage for data persistence.

- Phase 4 : Testing and Documentation

The final project phase was dedicated to quality insurance. We all tested the application in our own way to replicate a typical user experience and possibly identify bugs or unexpected behavior in our software. We also used this time to update the documentation and clean up the Github from previous test files that were no longer relevant.

5.2 Team Collaboration

To ensure efficient teamwork and code collaboration, we used industry-standard tools :

We utilized Github to share our progress, enabling everyone to test the software and implement new features or modifications. As communication is key in such a large group, we immediately created a Teams conversation to discuss challenges, track progress and distribute tasks. At times we organized face-to-face meeting to ensure a clearer and faster feedback.

Finally, we selected Overleaf for drafting the technical documentation and this report, while Powerpoint was used for the presentation support.

Let us now detail the roles and responsibilities. While all members contributed to the codebase, specific responsibilities were assigned to leverage individual strengths and account for weaknesses :

Name	**Role**
Mathieu Legrand	Search
Eliott Moskowicz	Favorites
Eva Dupin	Itinerary
Thomas Pestel	Budget
Maysa Folliet	Profile
William Publicol	Review
Paul Simon Vorderbruegge	Escape & Initialization
Florian Dorchies	History
Ivan Peterschmitt	Map Integration
Juliette Duclosson	HTML/CSS (Front-End 1)
Nahuel Bar Estrada	HTML/CSS (Front-End 2)
Etienne Fontbonne	DB & Debugging

Table 1: Team Members & Assigned Roles

Beyond their core development roles such as Search, Favorites, Budget, and Map Integration each team member actively participates in the creation and finalization of the project documentation :

Every team member contributed to the writing of the final report. This ensured that the documentation accurately reflects the planning, design, implementation, and testing phases they experienced. This approach guarantees subject matter expertise for each documented feature.

On top of that, a small team was also responsible for formatting and structuring the GitHub repository. This task involves writing a clear README file, making sure any outside user could understand the installation guide as well as updating the issue board.

6 Project Conclusion

The goal of this project was developing a travel-planning web application using only JavaScript, HTML and CSS in order to keep the project accessible for team members with little programming experience. The idea was to keep the website simple and intuitive while providing all the functionality needed for travel-planning.

We successfully implemented all major features outlined in our SRS, including destination search, itinerary management, user profiles, authentication and budget management, all of which was embedded into an intuitive and clean UI. Through extensive documentation we kept everything organized and recorded our work.

Because team-members had varying degrees of experience using git and github an early challenge was establishing efficient collaborative workflows. It was initially difficult to coordinate tasks and efficiently debug code but we improved our communication and team-work and managed to overcome these problems. On a technical level we struggled with persistent data storage which we managed to fix by implementing LocalStorage.

The project provided all of us with great hands-on experience in developing a web application. We learned how to collaborate using github and improved both our communication and teamwork. On the technical side we became more comfortable with JavaScript and handling persistent data storage using LocalStorage. The deeper we got into the implementation the more we appreciated the time we had spent planning and organizing the project in the beginning. This really helped us realize the advantages of a structured development process.

Overall we managed to deliver a web application that met our objectives of a clean and user-friendly travel-planner. Despite our initially limited experience we managed to collaborate effectively, implement all major features and produce clear documentation. The final outcome highlights how much we were able to learn and improve over the course of the development process.