APPLIED INFORMATION TECHNOLOGY AND COMPUTER SCIENCE
e-ISSN: 2773-5141

AITCS

Vol. 5 No. 2 (2024) 480-499
https://publisher.uthm.edu.my/periodicals/index.php/aitcs

Trek: Real-Time Collaborative Travel Planner Mobile Application

Wong Car Sing[1], Norfaradilla Wahid[2*]

[1,2] *Fakulti Sains Komputer dan Teknologi Maklumat,*
Universiti Tun Hussein Onn Malaysia, Parit Raja, Batu Pahat, 86400, MALAYSIA

*Corresponding Author: faradila@uthm.edu.my
DOI: https://doi.org/10.30880/aitcs.2024.05.02.026

Article Info

Received: 13 June 2024
Accepted: 28 September 2024
Available online: 15 December 2024

Keywords

Travel planning, mobile application. real-time collaboration, trip organization, collaborative, collaboration, itinerary, budgeting, checklist, reminder, document storage.

Abstract

This project develops Trek, a real-time collaborative travel planning mobile application that tackles the fragmented and inefficient nature of current trip planning tools, leading to smoother group coordination. Existing tools lack capabilities for real-time multi-user planning and updates, resulting in duplicated efforts, scheduling conflicts, and incomplete information sharing among groups planning trips together. Employing an incremental development model, iterative test-driven development, and technologies such as Flutter, Dart, MySQL and PHP, Trek focuses first on streamlined collaborative itinerary creation via mobile devices. The completed mobile application provides a comprehensive toolkit that simplifies travel organization for both individuals and groups, with features like budgeting, checklists, reminders, and document storage empowering seamless real-time collaboration. User testing yielded an average rating of 4.77 out of 5 stars, demonstrating Trek's effectiveness in enhancing travel planning workflows. Future work may explore performance optimization for high user loads, offline capabilities, integration with travel booking services, and leveraging AI for personalized recommendations.

1. Introduction

Travel planning is a complex process that requires coordinating transportation, lodging, activities, and other logistics. Despite the rapid growth of travel applications globally, many travel applications have received poor reviews for not meeting user expectations in various regions, including China, rendering them unable to generate desired user responses [1]. Currently, group travel planning relies on a fragmented workflow, with team members emailing documents back and forth, manually updating copies of itineraries, and having out of sync schedules across multiple platforms. This leads to duplicated work, delays, and frustration. This paper proposes Trek, a mobile travel assistant that consolidates essential travel organization tools into one platform. Trek not only integrates trip planning functionality into a single application, but also enables real-time simultaneous editing of itineraries, budgets and more across a group.

The problem Trek addresses is the fragmentation of current travel applications into separate tools for itinerary building, budgeting and so on. The fragmentation causes confusion and inefficiency for travellers trying to plan trips, especially groups coordinating schedules [2]. There is a need for a unified toolkit that streamlines the entire travel workflow.

The objectives of this study are:

i. To analyse and design an application that enables travellers to efficiently manage their trips, whether it is a single expedition or a group outing, before, during, or after the trip.

ii. To develop a practical and useful travel planning mobile application designed for individuals who are relatively new to or have intermediate experience in travel planning.
iii. To evaluate the functionality and user-friendliness of the created application.

The expected outcome is an application that simplifies organization for travellers by centralizing scattered resources. The key value offered by Trek is providing an all-in-one travel toolkit on mobile devices, saving travellers from piecing together various sources. With real-time collaboration, financial tracking, and other integrated supports, Trek aims to fundamentally improve independent and group travel planning.

The paper is organized as follows: Section 1 introduces the Trek application. Section 2 reviews related work. Section 3 outlines the methodology. Section 4 presents results and discussions. Section 5 concludes the project.

2. Related Work

Comparative studies were conducted on three existing travel applications – TripIt [3], RoadTrippers [4] and SaveTrip [5]. Each platform serves particular needs in the travel planning process, but has limitations in providing an integrated solution.

TripIt specializes in automated itinerary creation from confirmation emails and delivering real-time alerts on delays, cancellations and so on. However, it lacks critical features like shared itinerary editing, route mapping, document storage, and budget tracking.

RoadTrippers focuses on interactive route planning, with customizable maps, turn-by-turn directions and recommendations for attractions along the way [6]. But it does not enable seamless collaboration and falls short on travel assistance capabilities.

SaveTrip aims to simplify the financial aspects of travel coordination, supporting multi-currency budgets, receipt logging and spend analytics. However, it does not allow co-editing of itineraries or access to guide content.

In contrast, Trek provides an end-to-end platform that synergizes the entire lifecycle of group travel planning and execution. Its unique strengths include real-time co-editing with robust access controls, integrated messaging, checklist, reminders and document storage; and versatile budget management tools. Trek addresses the limitations of existing point solutions through its comprehensive feature set. Table 1 shows the comparison between proposed application and existing application.

Table 1 *Comparison between Proposed Application and Related Work*

System / Feature	TripIt	RoadTrippers	SaveTrip	Trek
Itinerary Creation	From emails and Fixed Templates	Fixed templates	Fixed templates	Customizable with rich formatting
Real-time collaboration	No	No	No	Yes
Itinerary sharing	Collaboration	Collaboration	Export Pdf	Collaboration
Budgeting tools	No	No	Multi- currency support	Expense tracking and monitoring
Document storage	No	No	Yes	Yes
Route planning	No	Interactive maps	Via Google Maps	Basic
Navigation	No	Turn-by-turn directions	No	No
Travel guides	No	Curated Guides	No	Basic
Checklists	No	No	Basic checklists	Custom checklists
Reminders	Booking alerts	Disruption alerts	No	Custom reminders
Platforms	iOS, Android	Website, iOS, Android	iOS, Android	iOS, Android

3. Methodology

This section describes the methodology and frameworks utilized in developing the Trek application. Fig. 1 illustrates the Incremental Model, highlighting various stages such as requirement analysis, design and development; testing and implementation. The incremental model was selected for the development of Trek application as it allows for iterative enhancements delivered in phases. This matched well with the objectives of

starting with core functionality like trip planning in initial releases, while expanding scope over multiple increments towards an integrated travel toolkit.

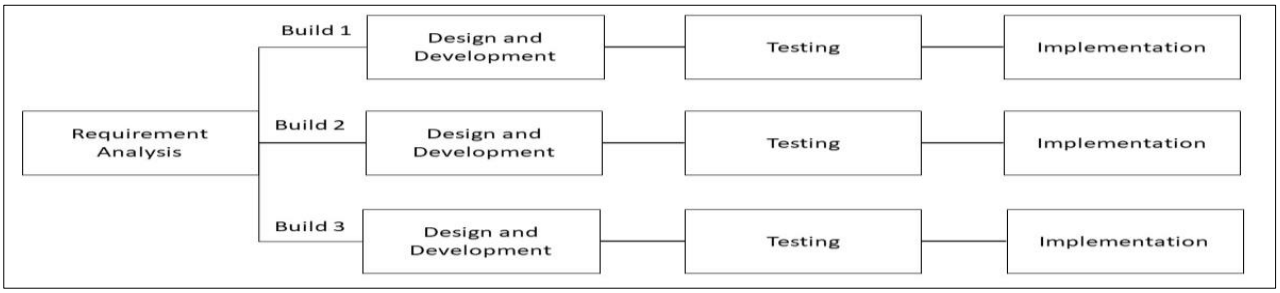


Fig. 1 *Incremental Model [7]*

Following the incremental model methodology, the system development of Trek application utilizes an incremental model comprising requirement analysis, design and development, testing, and implementation phases recurring in a cyclic approach. The Gantt Chart constructed for this project according to this incremental model is shown in Appendix A.

3.1 Requirement Analysis

The initial requirements analysis phase for the Trek application includes two methods. First, user opinions were gathered through questionnaires to understand their needs and pain points. This process involved documenting detailed specifications that outlined the system's capabilities and limitations, workflow diagrams for key functionalities, and potential interface layouts.

Second, a thorough examination of existing travel planning solutions and processes was conducted. This involved capturing both the functional needs and non-functional expectations of potential users. Extensive market research, including surveys across diverse demographics and competitor evaluations, identified key areas for improvement. These included opportunities in itinerary building, group trip coordination, budget integration, and document consolidation.

The initial analysis provides a solid foundation, but user needs and project realities might evolve. To ensure the Trek application remains aligned with these changes, iterative requirements analysis approach is employed. This involves revisiting and refining the initial deliverables throughout the development process. The next subsections present the some of the deliverables during this phase.

3.1.1 Functional and Non-functional Requirements

Both functional and non-functional expectations were gathered during initial planning activities. Detailed specifications were created in the analysis phase spanning system capabilities, constraints, workflows and layouts.

Functional requirements centred on end-user actions like registering accounts, building multi-day itineraries, creating reminders and checklists, uploading documents, and tracking expenses for trips. A few important functional requirements include:

i. The system should be able to allow users to create comprehensive trip plans, incorporating details from various modules, including itinerary, budgets, expenses, documents, checklists, and reminders.
ii. The system should be able to allow users to create new trip activity plans by inputting details such as activity name, time, and description of the activity.
iii. The system should be able to allow trip owners to invite others to collaborate on the planning and execution of trips and more.

Non-functional criteria examined include usability, where the system interface must be intuitive and user-friendly. Performance is evaluated, ensuring that the system responds quickly to user requests. Additionally, availability is addressed, aiming for key system functions to have high availability with minimal downtime. Security is examined to ensure a high level of protection for user account information and data. Furthermore, privacy is safeguarded, ensuring that user account information is not allowed to be shared without explicit consent.

3.1.2 Use Case Diagram

The use case diagram models the major capabilities of the system from an end user perspective. It identifies the primary external actors interacting with the system and the core use cases they perform. Fig. 2 shows an overview use case diagram for the Trek application.

Applied Information Technology and Computer Science Vol. 5 No. 2 (2024) p. 480-499

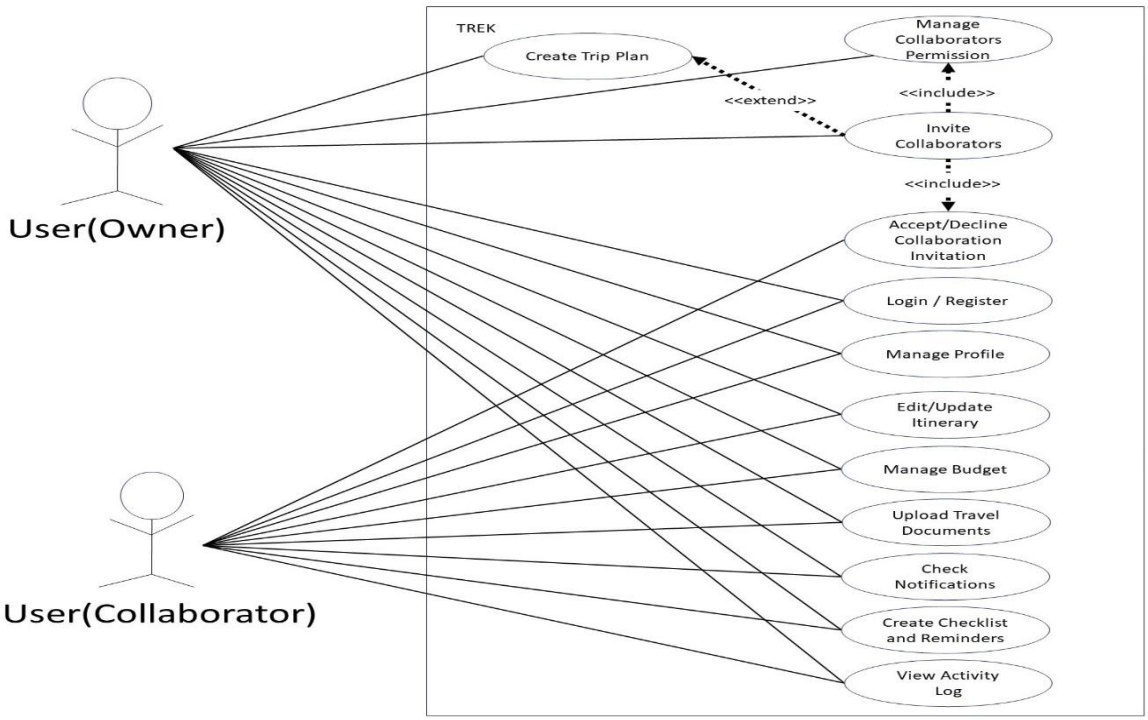


Fig. 2 *Use Case Diagram*

The Trek application revolves around two personas: Owners, who create and manage trip plans, and Collaborators, who assist in planning. Owners are able to create trip plans, invite collaborators and manage collaborators permission. Collaborators are able to accept or decline invitations. All users are able to login or register, manage profile, edit or update itinerary, manage budget, upload travel documents, check notification, create checklist and reminders; and view activity log.

3.1.3 Sequence Diagrams

The Adding Trip Plan Sequence Diagram in Fig. 3 outlines the process of creating a new trip itinerary in the system. The user initiates the sequence by selecting the option to add a trip, entering details through a form, including destination and budget. The information is processed and stored using various methods like *_handleCreateNewTripPlan().* Collaborators are able to be added, and the new trip is saved to the database. A success confirmation is displayed, and the user's updated trip list is shown.

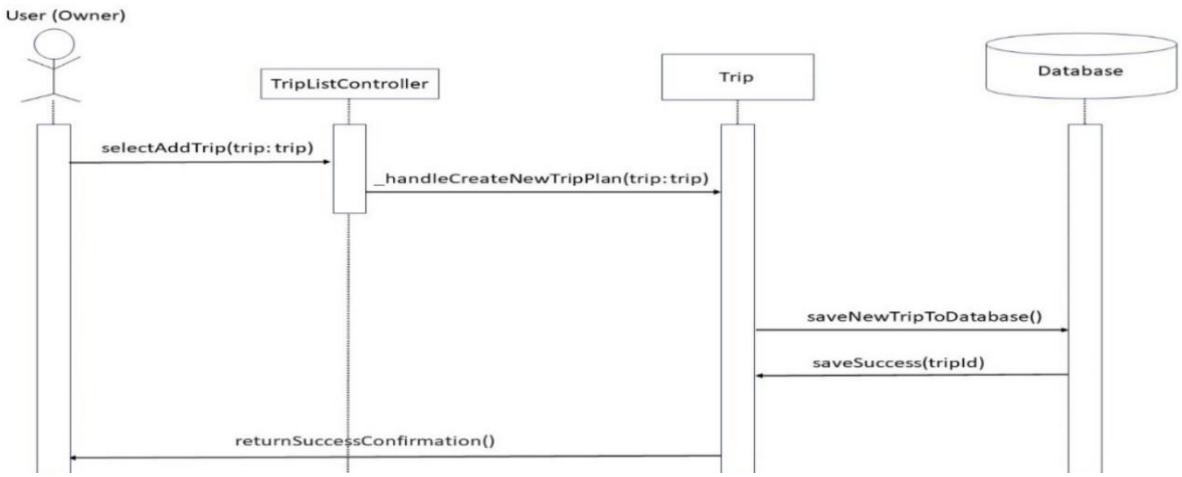


Fig. 3 *Adding Trip Plan Sequence Diagram*

The Invite Collaborators Sequence Diagram in Fig. 4 outlines the steps for a trip owner to invite users to collaborate on a trip. The Owner initiates the sequence by selecting to add a collaborator, enters invite details, and triggers an email invite. The Activity Log and Database are updated, and a notification is sent to the collaborator. The collaborator, upon receiving the invite, is able to choose to accept or decline.

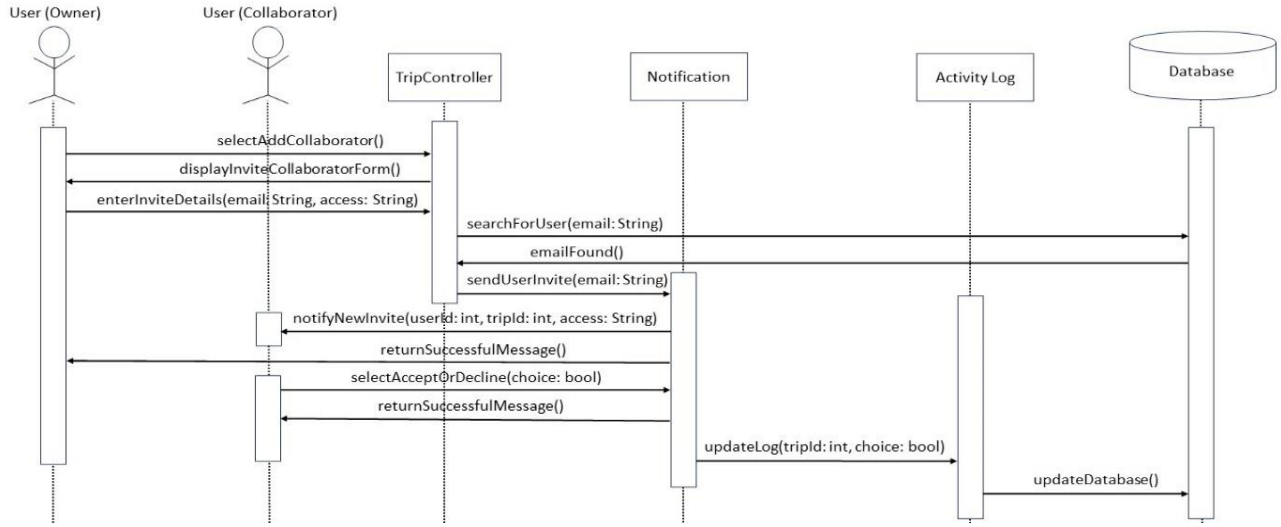


Fig. 4 *Invite Collaborators Sequence Diagram*

3.1.4 Activity Diagram

The activity diagram depicts user interactions within the travel application after launch. This includes functionalities like login, registration, password reset, and navigating to the homepage upon successful login. The home page serves as a central hub, offering users the ability to view the main activity log and create or view trip plans. The Itinerary module allows users to create new activity plans, update existing ones, and manage the expenses for each day within the itinerary. The Budget module manages finances, and the Documents module facilitates file-related tasks. Checklists & Reminders enable users to create and track lists and to-do items. The Activity Log provides a chronological overview of significant trip plan changes. Appendix B, Fig. B1 shows the Activity Diagram.

3.1.5 Class Diagram

The class diagram models the design of the proposed Trek application and has thirteen key classes related to trip management, excluding the User-Module Controllers. The Trip class is at the center, responsible for managing instances of other trip-related classes, including Reminder, Budget, Itinerary, Document, and Checklist for a particular trip. The class diagram is able to be seen in Appendix B, Fig. B2.

3.2 Design

With the high-level specifications established, initial design commenced by modelling the system architecture, database schema, wireframes, and other structural components essential for translating specifications into eventual working code. A multi-tier architecture promotes separation of concerns across the presentation, business logic and data access layers for easier maintainability down the line. Relational database techniques allow flexibility to extend entity relationships over time. The wireframes visualize critical workflows for user signups, trip creation, itinerary building, and budget tracking informed by workflow analyses. The next subsections present the deliverables during the design phase. Additional major deliverables of this phase is shown in Appendix C.

3.2.1 Data Models

The data dictionary provides the metadata definitions for the database structure and contents for the Trek application. It documents tables, column attributes along with their properties and relationships with other database entities. The data dictionary designed for Trek application include TripController and Collaborations Data Dictionary shown in Table 2 and Table 3 respectively.

Table 2 *TripController Data Dictionary*

No	Attribute	Data Type	Size	Key	Description
1	tripId	int	-	PK	Unique ID of trip
2	name	varchar	255	-	Name of trip
3	destination	varchar	255	-	Trip location
4	startDate	date	-	-	Start date
5	endDate	date	-	-	End

Table 3 *Collaborations Data Dictionary*

No	Attribute	Data Type	Size	Key	Description
1	collaborationId	int	11	PK	Unique ID of collaboration
2	tripId	int	11	FK	ID of the trip this collaboration belongs to
3	userId	int	11	FK	ID of the user
4	accessLevel	varchar	50	-	Access level of user
5	accepted	tinyint	1	-	Collaboration invitation status

TripController data dictionary stores trip-related details such as the trip ID, name, destination, start date, and end date. The Collaborations data dictionary is responsible for storing information related to collaborators involved in a specific trip. It allows the application to track which users have been invited to participate in a specific trip and what level of access they have been granted. This information enables the application to enforce access control, ensuring that only authorized collaborators are able to view or modify trip details based on their assigned access level.

3.2.2 User Interface Design

Good user interface design is essential for effective interaction. Appendix C.2, Fig. C1 and Fig. C2 show the different modules of the Trip Plan feature in the Trek application, providing a visual representation of the user interface for key components such as Document Module, Itinerary Module and Budget Module.

3.3 Development

Following the groundwork of design structures, the frontend was built using Flutter, which allowed crafting high-quality native interfaces for both Android and iOS platforms from a single codebase. State management principles organized code modularly enhancing cohesion. An animation framework provided smooth transitions realizing the envisioned user experience.

The backend was developed using Dart and Flutter, leveraging the same language for crafting both the frontend and backend code. This enabled code reuse and synchronized updates between the layers. MySQL database was integrated through the use of PHP to enable persistent storage and efficient retrieval of travel data. ORM techniques abstracted query logic behind clean repository interfaces providing object abstractions for entities like trips, users and so on.

3.4 Testing

Initially, a basic system prototype is developed to provide end-users an early opportunity to interact with core functionalities like user registration, login, and basic trip planning workflows. This initial prototype focuses solely on these critical features, enabling the gathering of user feedback regarding the intuitiveness and usability of these fundamental workflows.

After testing the core modules in the initial prototype, the testing process continues incrementally with each subsequent iteration. As additional features are introduced, such as budget tracking, reminders, document sharing, and more, each new capability undergoes thorough testing. This testing phase occurs after the design and development of each incremental feature set. Functional testing is conducted on each new code module to validate its functionality at the component level, ensuring it works as expected.

3.5 System Development Workflow

The system development of Trek application utilizes an incremental model comprising requirement analysis, design and development, testing, and implementation phases recurring in a cyclic approach. Table 4 is a summary table outlining the phases, activities, and deliverables for the incremental development process:

Table 4 *Summary Table for the Incremental Development Process*

Phase	Activities	Deliverables
Initial Requirement Analysis	• Requirements gathering through surveys of travellers to understand challenges faced in trip planning processes. • Competitor analysis examining limitations of existing travel apps like TripIt, RoadTrippers and SaveTrip.	• User survey results on trip planning challenges. • Competitor analysis report. • Prioritized feature list focused on real-time collaboration, budgeting and more.

Table 4 *(cont.)*

Phase	Activities	Deliverables
Initial Requirement Analysis	• Persona modelling based on identified user groups and their planning needs. • Workflow analysis mapping out collaborative trip planning steps • Definition of high-level specifications and key features.	• Development increment plan aligned to core modules. • Project timeline and milestones.
Subsequent /Iterative Requirement Analysis	• Expand specifications for real-time collaboration and more. • Refine use cases for trip planning workflows, including scenarios with multiple collaborators. • Update sequence diagrams to reflect real-time updates.	• Expanded specifications for real-time collaboration. • Refined use cases and sequence diagrams for collaborative planning workflows.
Design	• Define 3-tier architecture with presentation, business and data layers suitable for travel app. • Build wireframes focused on key workflows like trip creation • Model MySQL database schema for trips, users, activities and so on, based on analysis phase entities.	• System architecture diagrams. • UI wireframes for key workflows. • Data dictionary and database schema.
Development	• Modular coding of core modules - user authentication, trip planning, itinerary building. • Implement persistence layer for MySQL tables like profile and trip details. • Develop Flutter front-end UI components.	• Source code for core modules like authentication, trip planning. • User interface code for key screens. • Database integration for persisting and retrieving trip plan data.
Testing	• Functional testing for individual code modules. like user authentication, trip creation, and expense tracking.	• Test case results and bug reports for each functional tested. • Bug fixes for issues identified during functional testing.
Implementation	• Integrate the tested and fixed code into the application. • Integration with the MySQL database to persist and retrieve application data.	• Stable increments of the application.
User Evaluation	• User acceptance testing with the target user groups to evaluate the application's usability and alignment with their expectations.	• User acceptance test results. • User feedback and prioritized enhancements.

4. Results and Discussions

The backend implementation of the Trek application is structured around a series of PHP scripts responsible for managing various functionalities and interactions with the database. Each PHP script corresponds to a specific aspect of the application's backend functionality such as authentication logic, trip plan creation logic and document storage logic. The mobile application's client-side implementation encompasses various functionalities, including user authentication, trip planning with collaboration features, file management, itinerary creation, budget management, checklists, reminders, and a user-friendly interface.

4.1 User Authentication Module

To handle user registration, login, and password reset, the application employs backend logic for authentication. Upon logging in successfully, a *Profile* class is created which stores the user's information, such as *userId*, *username*, and *email*. This profile data is fetched from the server using the *fetchUserData* method in the *UserManager* class. Upon a successful server response, the application parses the user's data and creates a *Profile* object to store the information. The process of creating and populating the *Profile* object is illustrated in Figures 5(a) and 5(b). The login screen is depicted in Fig. 6(a) and successful login redirects the user to the homepage shown in Fig. 6(b) which features the My Trips screen, shown in Fig. 6(c), accessible through the bottom navigation bar.

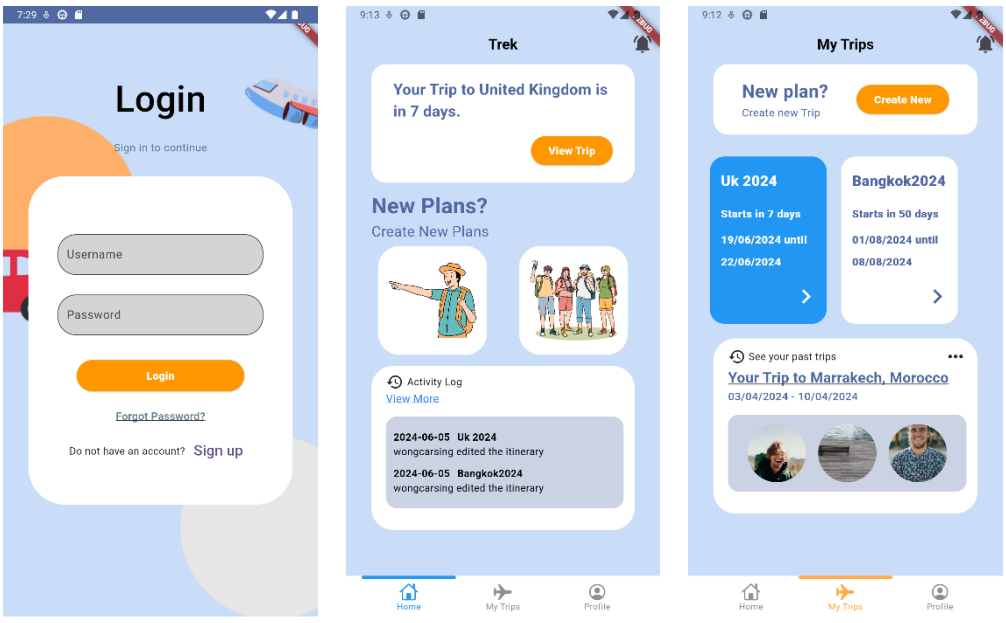
```dart
void _handleLogin() async {
  try {
    // Retrieve user input
    final username = _usernameController.text;
    final password = _passwordController.text;

    // Validate input (e.g., check for empty fields, password strength)
    if (username.isEmpty || password.isEmpty) {
      // Display error message for missing fields
      showSnackbar(
          context, 'Username and password are required.', Colors.red);
      return;
    }

    // Make POST request to auth.php
    final response = await http.post(
      Uri.parse('$BASE_IP/flutter_api/auth.php'),
      body: {
        'login': 'true',
        'username': username,
        'password': password,
      },
    );

    // Handle response
    final responseData = jsonDecode(response.body);
    print(responseData);
    if (responseData['success']) {
      await UserManager.setLoggedInUser(username);
```

(a)

```dart
static Future<Profile?> fetchUserData(String username) async {
  try {
    final response = await http.post(
      Uri.parse('$BASE_IP/flutter_api/auth.php?'),
      body: {'fetchUserData': 'true', 'username': username},
    );

    if (response.statusCode == 200) {
      final responseData = jsonDecode(response.body);

      if (responseData['success'] ?? false) {
        final userData = responseData['user_data'] as Map<String, dynamic>?;

        if (userData != null) {
          loggedInUserProfile = Profile.fromMap(userData);
          print(loggedInUserProfile);
          return loggedInUserProfile;
        } else {
          print('Failed to fetch user data. User data is null.');
          return null;
        }
      } else {
        print('Failed to fetch user data. Server response: $responseData');
        return null;
      }
    } else {
      print('Failed to fetch user data. Status code: ${response.statusCode}');
      return null;
    }
  } catch (error) {
    print('An error occurred while fetching user data: $error');
    return null;
  }
```

(b)

Fig. 5 *Login Logic Code Segment (a) Sending Login Request; (b) Receiving User Data*



(a) **(b)** **(c)**

Fig. 6 *Initial Screens (a) Login; (b) Homepage; (c) My Trips*

4.2 Key User Interfaces

This section outlines the key user interfaces within the application: All Activity Log of all trip plans, Trip Dashboard of a specific trip plan and the Manage Collaborator View of a trip plan. The design prioritizes user-friendliness with intuitive navigation and clear information architecture. Through a visually appealing and well-organized layout, the interface aims to facilitate effortless interaction for users of all experience levels. Fig 7 shows the screens for All Activity Log, Trip Dashboard and Manage Collaborator View respectively.

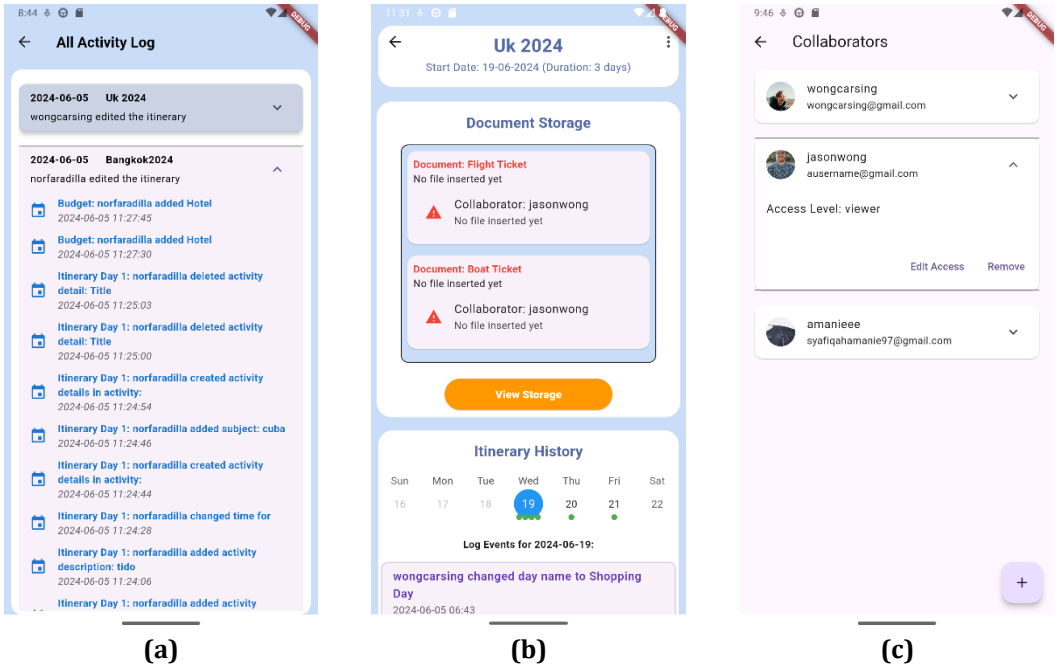


Fig. 7 *Key User Interfaces (a) All Activity Log; (b) Trip Dashboard; (c) Manage Collaborator View*

4.3 Trip Plan Creation Module

The application offers two trip plan creation options: solo travel for individual needs, and group trips designed for collaborative planning. As illustrated in Fig. 7(c), trip owners for group trips are able to manage collaborators, assign permissions, and track their progress.

When creating a group trip plan, users input details like trip name, destination, budget, dates, currency, and collaborator lists. The application validates this input to ensure data integrity and consistency. The backend then verifies the existence of essential data. Upon successful validation, the trip details are then inserted into the database, followed by the addition of the trip owner to the collaboration list. Activity logs are updated for both the owner and collaborators. Collaborators' details are iteratively added to the *collaborations* table. Subsequently, itinerary dates are inserted, and budgets are calculated and assigned to both the owner and collaborators. Upon successful execution, the *tripId* is returned.

The application employs robust access control and permission management for collaborators. The owner is able to set collaborators as viewers or editors - viewers are able to only view the itinerary plan, while editors are able to modify details of the itinerary. Owners have special privileges to add required expenses for collaborators, create document upload slots that collaborators must complete, and assign checklist items. The document module allows owners to track each collaborator's progress on uploading required files. Activity logs meticulously track all user actions, such as itinerary changes and file uploads, keeping everyone informed about updates made by other collaborators. This transparency eliminates coordination conflicts that plague disjointed workflows involving emails and messages. The owner is able to invite new collaborators at any time, who have the ability to accept or decline the invitation. Accepted collaborators gain access based on their assigned permissions.

Conversely, for solo trip plan creation, similar validation checks are conducted. Upon successful validation, trip details are inserted into the database. The owner is added to the collaboration list, and activity logs are updated accordingly. Itinerary dates are inserted, and budgets are calculated and assigned to the owner. Upon successful execution, the *tripId* is returned, finalizing the creation process.

Applied Information Technology and Computer Science Vol. 5 No. 2 (2024) p. 480-499

4.4 Itinerary, Budget and Document Module

The Itinerary module allows users to meticulously plan and organize their travel activities. The Budget module seamlessly integrates with the Itinerary module, enabling users to effectively plan and track their expenses throughout the trip. Users are able to define an overall budget for their trip, then allocate specific amounts to various categories like accommodation, transportation, and activities. Additionally, the system allows for detailed expense tracking by associating individual costs with specific days throughout the trip. The Document Storage module provides a secure, local way to organize and access trip-related documents like flight tickets and vouchers. Trip owners are able to create mandatory document slots for collaborators and track their progress in uploading the required files.

The backend logic that handles logging expenses, creating activities and creating trip file slots is shown in Appendix D, Fig. D1, Fig. D2 and Fig. D3 respectively. When adding an activity to the itinerary, the backend verifies the essential parameters, constructs an SQL query to insert the activity into the database, and associates it with the specified itinerary.

For budget management, the backend fetches existing budget details, including overall budget, daily budgets, and associated expense items, by executing SQL queries. Users are able to create new expense items, which are then associated with the corresponding daily budget. If collaborators are involved, the backend processes each collaborator to add the same expense item to their respective budgets.

The Document Storage module enables creating trip file slots, where users are able to upload and store essential documents. The backend validates the incoming request, inserts a new slot into the database, and generates a response with relevant slot information. If collaborators are specified, the backend creates additional slots for each collaborator, allowing them to upload files while collectively tracking their progress. Fig. 8 shows the interfaces for Itinerary Module, Budget Module and Document Module.

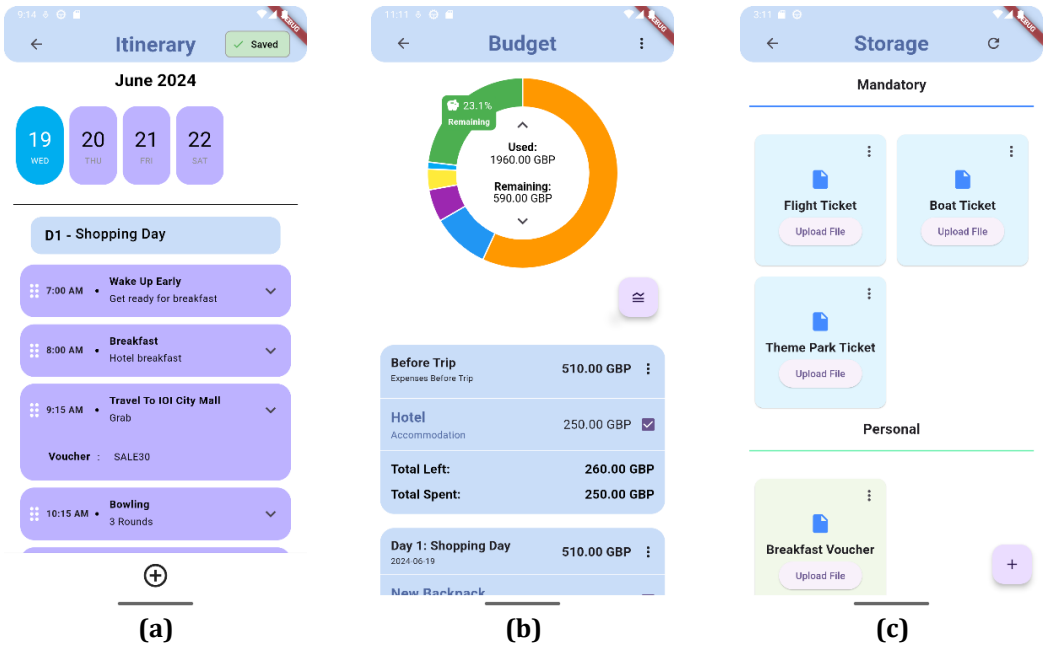


Fig. 8 *Modules Interfaces (a) Itinerary; (b) Budget; (c) Document*

4.5 Checklist, Reminder and Activity Log Module

The Checklist and Reminder modules are designed to help users stay organized and ensure they do not overlook essential tasks and events during their trip planning and execution. The Checklist module allows users to create and manage customized checklists, streamlining the preparation process and reducing the risk of overlooking critical details. The Reminder module enables users to create customized reminders, helping them effectively plan and manage their schedules.

The backend logic for creating a new checklist involves processing the request, inserting a new checklist entry into the database, and retrieving the new checklist ID. If it is a group trip, the backend processes collaborator slots and creates additional checklist entries for each collaborator, allowing them to have their own checklists associated with the same trip. This logic is shown in Appendix D, Fig. D4.

Similarly, for creating a new reminder, the backend processes the request, inserts a new reminder into the database, and retrieves the new reminder ID. This ID is used for future actions, such as updating the notification status when the app successfully sets the reminder.

The Activity Log module maintains a record of events, such as changes to the itinerary, new required documents added by the trip owner, and other relevant activities. Fig. 9 shows the interfaces for Checklist Module, Reminder Module and Activity Log Module.

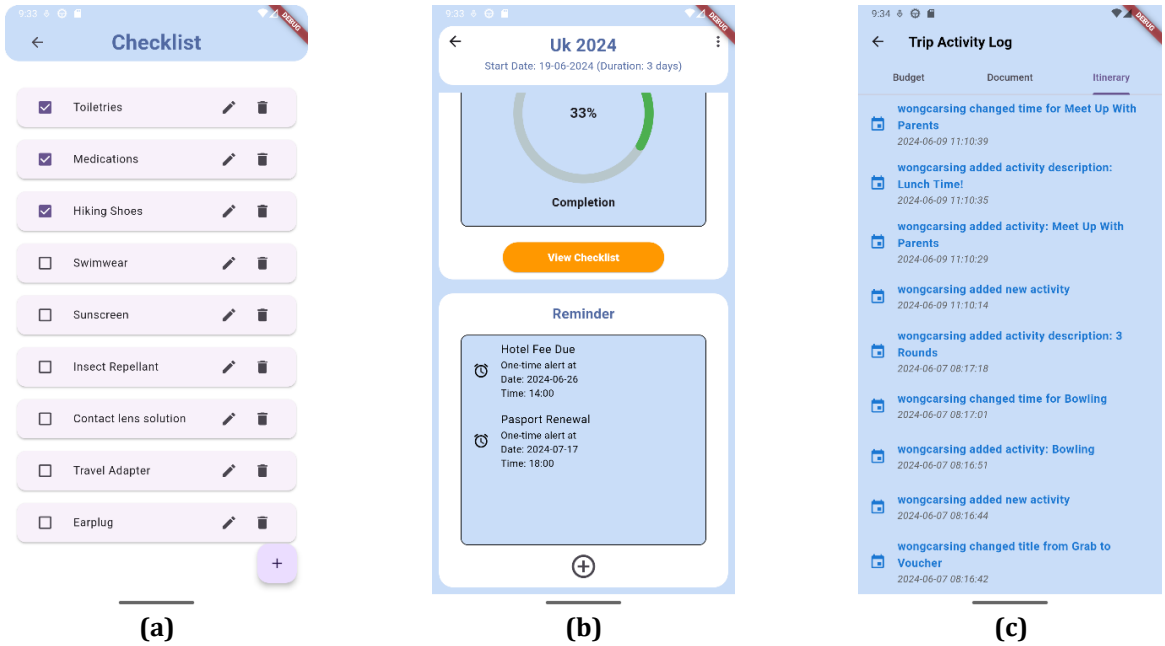


Fig. 9 *Modules Interfaces (a) Checklist; (b) Reminder; (c) Activity Log*

4.6 Testing

System testing ensures that all components of the Trek Travel Planner application work as intended and meet the requirements specified in the design. This section covers functional testing and user acceptance testing (UAT) for the system. The major deliverables of this phase are shown in Appendix E.

4.6.1 Functional Testing

Functional testing involves testing each individual component or module in isolation to ensure they function correctly. The functional testing covered various modules of the application, with the test case results documented in the following tables in Appendix E:

- Table E1: Test Case Results for Budgets & Expenses Management Module
- Table E2: Test Case Results for Trip Plan Management Module
- Table E3: Test Case Results for Documents Management Module
- Table E4: Test Case Results for Collaborators Management Module
- Table E5: Test Case Results for Trip Itinerary Management Module

The results across all these modules were satisfactory, with no major issues or user-faced problems identified during the functional testing phase. All the test cases were successfully passed, validating that each module met the specified requirements and functioned as per the design specifications when tested independently.

4.6.2 User Acceptance Testing (UAT)

User acceptance testing (UAT) was conducted to verify that the system meets the needs of end users and functions effectively in real-world scenarios. A diverse group of participants participated in the UAT, representing various age groups, genders, and travel frequencies (from infrequent to very frequent travelers). Their primary travel purposes were leisure and a combination of leisure and business. The functionalities tested during UAT are detailed in Appendix F, Fig. F1 and the application received a rating of 4.77 out of 5 stars from the users.

5. Conclusions

The aim of this work is to develop Trek, an integrated mobile travel planner application that streamlines and simplifies trip planning and organization for both individuals and groups. The problems addressed include the

fragmentation of existing tools causing inefficient workflows, scheduling conflicts due to out-of-sync plans across users, and the lack of seamless collaboration capabilities.

Following an incremental development methodology, the project successfully developed and tested the Trek real-time collaborative travel planner mobile application. The application integrates various functionalities such as user authentication, trip planning, budgeting, document management, checklists, reminders, and collaborative editing. The backend was implemented using PHP, and the client-side was developed using Flutter and Dart. The application received positive feedback during user acceptance testing, with an average rating of 4.77 out of 5 stars.

Despite its success, the project has several limitations. The application's real-time collaboration feature may experience latency issues with a large number of simultaneous users. The current implementation is also dependent on a stable internet connection, which may not always be available to all users. Additionally, the application's functionality is focused primarily on travel planning, and it may not cater to other related aspects such as travel bookings or local guides. Furthermore, the document management feature only saves the file path and not the actual PDF file itself.

Future developments could address the limitations mentioned. Enhancing the application's performance under high user load and improving offline capabilities could significantly increase its usability. Expanding the application to include features like travel bookings, local guides, and integration with other travel services would make it a more comprehensive travel solution. Incorporating advanced technologies such as artificial intelligence for personalized recommendations and augmented reality for enhanced travel experiences could also be explored.

Acknowledgement

The authors would like to thank the Faculty of Computer Science and Information Technology, Universiti Tun Hussein Onn Malaysia for its support.

Conflict of Interest

Authors declare that there is no conflict of interests regarding the publication of the paper.

Author Contribution

This journal requires that all authors take public responsibility for the content of the work submitted for review. The contributions of all authors must be described in the following manner:

The authors confirm contribution to the paper as follows: ***study conception and design:*** *Wong Car Sing, Dr. Norfaradilla Wahid;* ***data collection:*** *Wong Car Sing, Dr. Norfaradilla Wahid;* ***analysis and interpretation of results:*** *Wong Car Sing, Dr. Norfaradilla Wahid;* ***draft manuscript preparation:*** *Wong Car Sing, Dr. Norfaradilla Wahid. All authors reviewed the results and approved the final version of the manuscript.*

References

[1] Z. Gao, J. H. Cheah, X. J. Lim, S. I. Ng, T. H. Cham, and C. L. Yee, "Can travel apps improve tourists' intentions? Investigating the drivers of Chinese gen Y users' experience," Journal of Vacation Marketing, vol. 30, no. 3, pp. 505-534, 2024.

[2] C. H. Chu and C. Huang, "A platform for travel planning by using Google maps," in 2015 16th IEEE Int. Conf. Mobile Data Management, vol. 2, pp. 120-125, Jun. 2015.

[3] D. Patel, "Seamless Group Trip Planning: Connecting Personal Inspirations to Unified Decisions," Ph.D. dissertation, Emily Carr Univ. Art and Design, Vancouver, BC, Canada, 2024.

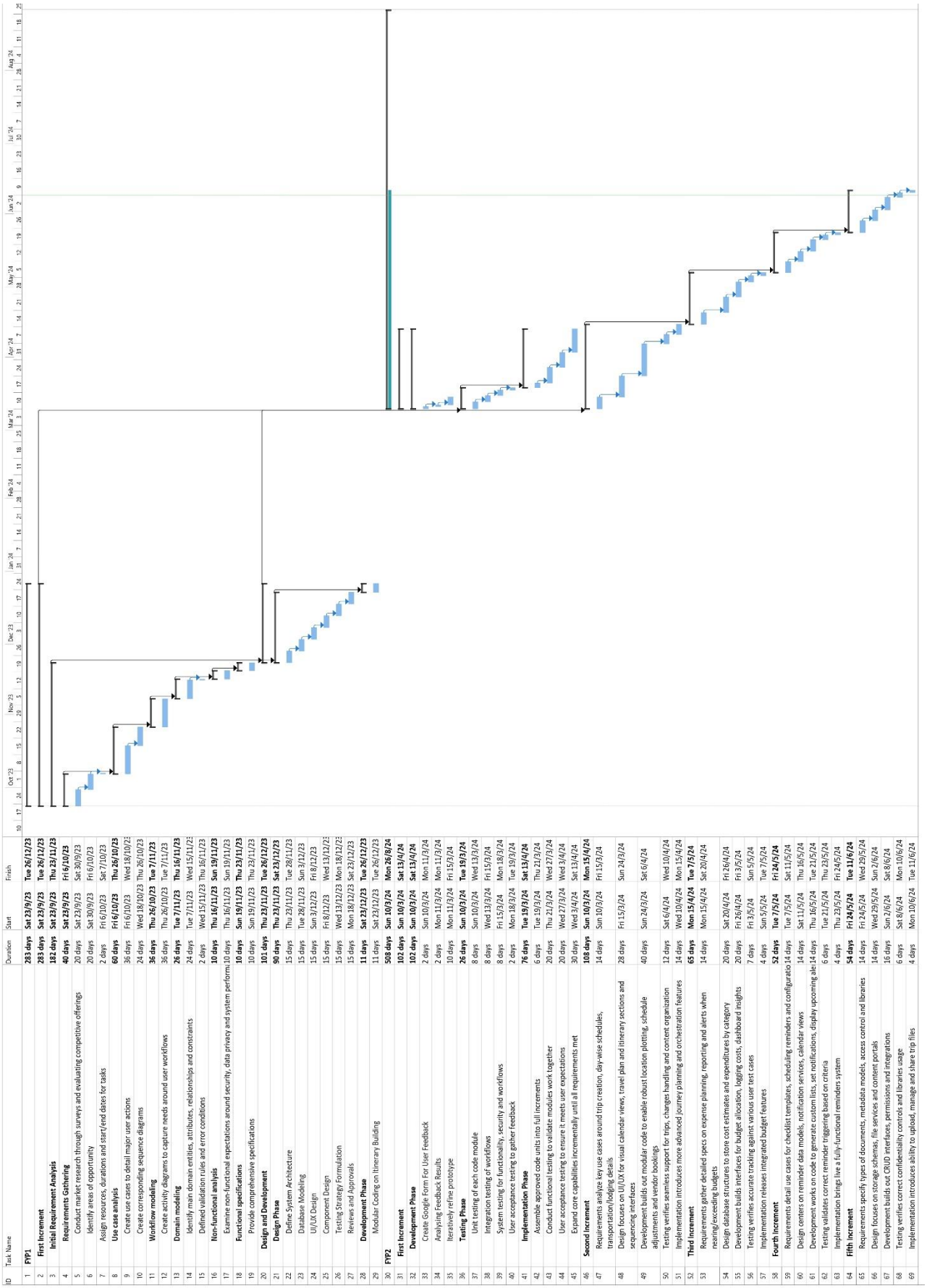
[4] Roadtrippers, "Roadtrippers - Trip Planner," Google Play Store, 2024. [Online]. Available: https://play.google.com/store/apps/details?id=com.roadtrippers. [Accessed: Nov. 11, 2023].

[5] D. Kim, "SaveTrip," Apple App Store, 2024. [Online]. Available: https://apps.apple.com/us/app/savetrip/id1500591552. [Accessed: Nov. 11, 2023].

[6] G. W. Tong, "The Development of Mobile Itinerary Planner for Kampar Tourism," Final Year Project, UTAR, 2021.

[7] S. S. Kute and S. D. Thorat, "A review on various software development life cycle (SDLC) models," International Journal of Research in Computer and Communication Technology, vol. 3, no. 7, pp. 778-779, 2014.

APPENDIX A: Gantt Chart



ID	Task Name	Duration	Start	Finish
1	**FYP1**	**283 days**	**Sat 23/9/23**	**Tue 26/12/23**
2	**First Increment**	**283 days**	**Sat 23/9/23**	**Tue 26/12/23**
3	**Initial Requirement Analysis**	**182 days**	**Sat 23/9/23**	**Thu 23/11/23**
4	**Requirements Gathering**	**40 days**	**Sat 23/9/23**	**Fri 6/10/23**
5	Conduct market research through surveys and evaluating competitive offerings	20 days	Sat 23/9/23	Sat 30/9/23
6	Identify areas of opportunity	20 days	Sat 30/9/23	Fri 6/10/23
7	Assign resources, durations and start/end dates for tasks	2 days	Fri 6/10/23	Sat 7/10/23
8	**Use case analysis**	**60 days**	**Fri 6/10/23**	**Thu 26/10/23**
9	Create use cases to detail major user actions	36 days	Fri 6/10/23	Wed 18/10/23
10	Create corresponding sequence diagrams	24 days	Wed 18/10/23	Thu 26/10/23
11	**Workflow modeling**	**36 days**	**Thu 26/10/23**	**Tue 7/11/23**
12	Create activity diagrams to capture needs around user workflows	36 days	Thu 26/10/23	Tue 7/11/23
13	**Domain modeling**	**26 days**	**Tue 7/11/23**	**Thu 16/11/23**
14	Identify main domain entities, attributes, relationships and constraints	24 days	Tue 7/11/23	Wed 15/11/23
15	Defined validation rules and error conditions	2 days	Wed 15/11/23	Thu 16/11/23
16	**Non-functional analysis**	**10 days**	**Thu 16/11/23**	**Sun 19/11/23**
17	Examine non-functional expectations around security, data privacy and system perform	10 days	Thu 16/11/23	Sun 19/11/23
18	**Functional specifications**	**10 days**	**Sun 19/11/23**	**Thu 23/11/23**
19	Provide comprehensive specifications	10 days	Sun 19/11/23	Thu 23/11/23
20	**Design and Development**	**101 days**	**Thu 23/11/23**	**Tue 26/12/23**
21	**Design Phase**	**90 days**	**Thu 23/11/23**	**Sat 23/12/23**
22	Define System Architecture	15 days	Thu 23/11/23	Tue 28/11/23
23	Database Modeling	15 days	Tue 28/11/23	Sun 3/12/23
24	UI/UX Design	15 days	Sun 3/12/23	Fri 8/12/23
25	Component Design	15 days	Fri 8/12/23	Wed 13/12/23
26	Testing Strategy Formulation	15 days	Wed 13/12/23	Mon 18/12/23
27	Reviews and Approvals	15 days	Mon 18/12/23	Sat 23/12/23
28	**Development Phase**	**11 days**	**Sat 23/12/23**	**Tue 26/12/23**
29	Modular Coding on Itinerary Building	11 days	Sat 23/12/23	Tue 26/12/23
30	**FYP2**	**508 days**	**Sun 10/3/24**	**Mon 26/8/24**
31	**First Increment**	**102 days**	**Sun 10/3/24**	**Sat 13/4/24**
32	**Development Phase**	**102 days**	**Sun 10/3/24**	**Sat 13/4/24**
33	Create Google Form For User Feedback	2 days	Sun 10/3/24	Mon 11/3/24
34	Analysing Feedback Results	2 days	Mon 11/3/24	Mon 11/3/24
35	Iteratively refine prototype	10 days	Mon 11/3/24	Fri 15/3/24
36	**Testing Phase**	**26 days**	**Sun 10/3/24**	**Tue 19/3/24**
37	Unit testing of each code module	8 days	Sun 10/3/24	Wed 13/3/24
38	Integration testing of workflows	8 days	Wed 13/3/24	Fri 15/3/24
39	System testing for functionality, security and workflows	8 days	Fri 15/3/24	Mon 18/3/24
40	User acceptance testing to gather feedback	2 days	Mon 18/3/24	Tue 19/3/24
41	**Implementation Phase**	**76 days**	**Tue 19/3/24**	**Sat 13/4/24**
42	Assemble approved code units into full increments	6 days	Tue 19/3/24	Thu 21/3/24
43	Conduct functional testing to validate modules work together	20 days	Thu 21/3/24	Wed 27/3/24
44	User acceptance testing to ensure it meets user expectations	20 days	Wed 27/3/24	Wed 3/4/24
45	Expand core capabilities incrementally until all requirements met	30 days	Wed 3/4/24	Sat 13/4/24
46	**Second Increment**	**108 days**	**Sun 10/3/24**	**Mon 15/4/24**
47	Requirements analyze key use cases around trip creation, day-wise schedules, transportation/lodging details	14 days	Sun 10/3/24	Fri 15/3/24
48	Design focuses on UI/UX for visual calendar views, travel plan and itinerary sections and sequencing interfaces	28 days	Fri 15/3/24	Sun 24/3/24
49	Development builds out modular code to enable robust location plotting, schedule adjustments and vendor bookings	40 days	Sun 24/3/24	Sat 6/4/24
50	Testing verifies seamless support for trips, changes handling and content organization	12 days	Sat 6/4/24	Wed 10/4/24
51	Implementation introduces more advanced journey planning and orchestration features	14 days	Wed 10/4/24	Mon 15/4/24
52	**Third Increment**	**65 days**	**Mon 15/4/24**	**Tue 7/5/24**
53	Requirements gather detailed specs on expense planning, reporting and alerts when nearing/exceeding budgets	14 days	Mon 15/4/24	Sat 20/4/24
54	Design database structures to store cost estimates and expenditures by category	20 days	Sat 20/4/24	Fri 26/4/24
55	Development builds interfaces for budget allocation, logging costs, dashboard insights	20 days	Fri 26/4/24	Fri 3/5/24
56	Testing verifies accurate tracking against various user test cases	7 days	Fri 3/5/24	Sun 5/5/24
57	Implementation releases integrated budget features	4 days	Sun 5/5/24	Tue 7/5/24
58	**Fourth Increment**	**52 days**	**Tue 7/5/24**	**Fri 24/5/24**
59	Requirements detail use cases for checklist templates, scheduling reminders and configuratio	14 days	Tue 7/5/24	Sat 11/5/24
60	Design centers on reminder data models, notification services, calendar views	14 days	Sat 11/5/24	Thu 16/5/24
61	Development works on code to generate custom lists, set notifications, display upcoming ale	14 days	Thu 16/5/24	Tue 21/5/24
62	Testing validates correct reminder triggering based on criteria	6 days	Tue 21/5/24	Thu 23/5/24
63	Implementation brings live a fully-functional reminders system	4 days	Thu 23/5/24	Fri 24/5/24
64	**Fifth Increment**	**54 days**	**Fri 24/5/24**	**Tue 11/6/24**
65	Requirements specify types of documents, metadata models, access control and libraries	14 days	Fri 24/5/24	Wed 29/5/24
66	Design focuses on storage schemas, file services and content portals	14 days	Wed 29/5/24	Sun 2/6/24
67	Development builds out CRUD interfaces, permissions and integrations	16 days	Sun 2/6/24	Sat 8/6/24
68	Testing verifies correct confidentiality controls and libraries usage	6 days	Sat 8/6/24	Mon 10/6/24
69	Implementation introduces ability to upload, manage and share trip files	4 days	Mon 10/6/24	Tue 11/6/24

Fig. A1 *Gantt Chart*

APPENDIX B: Deliverables during Requirement Analysis Phase

B.1 Activity Diagram

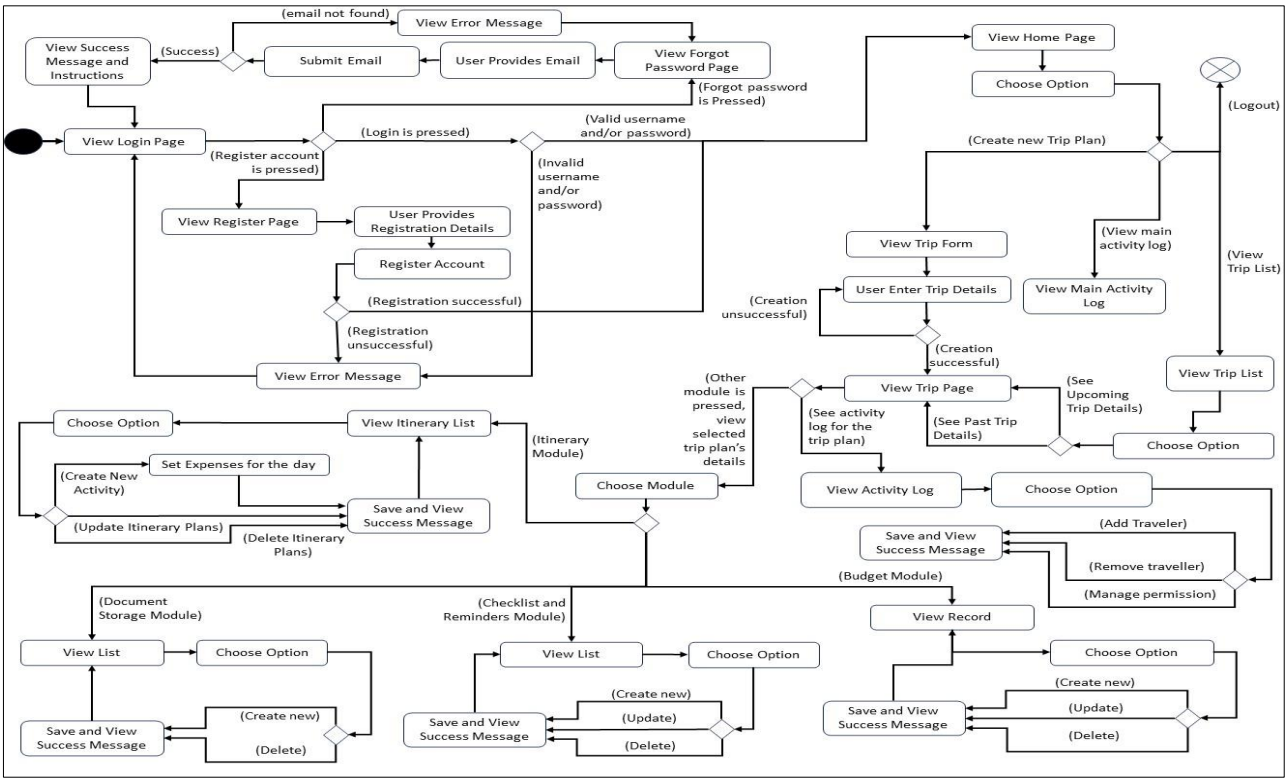


Fig. B1 *Activity Diagram*

B.2 Class Diagram

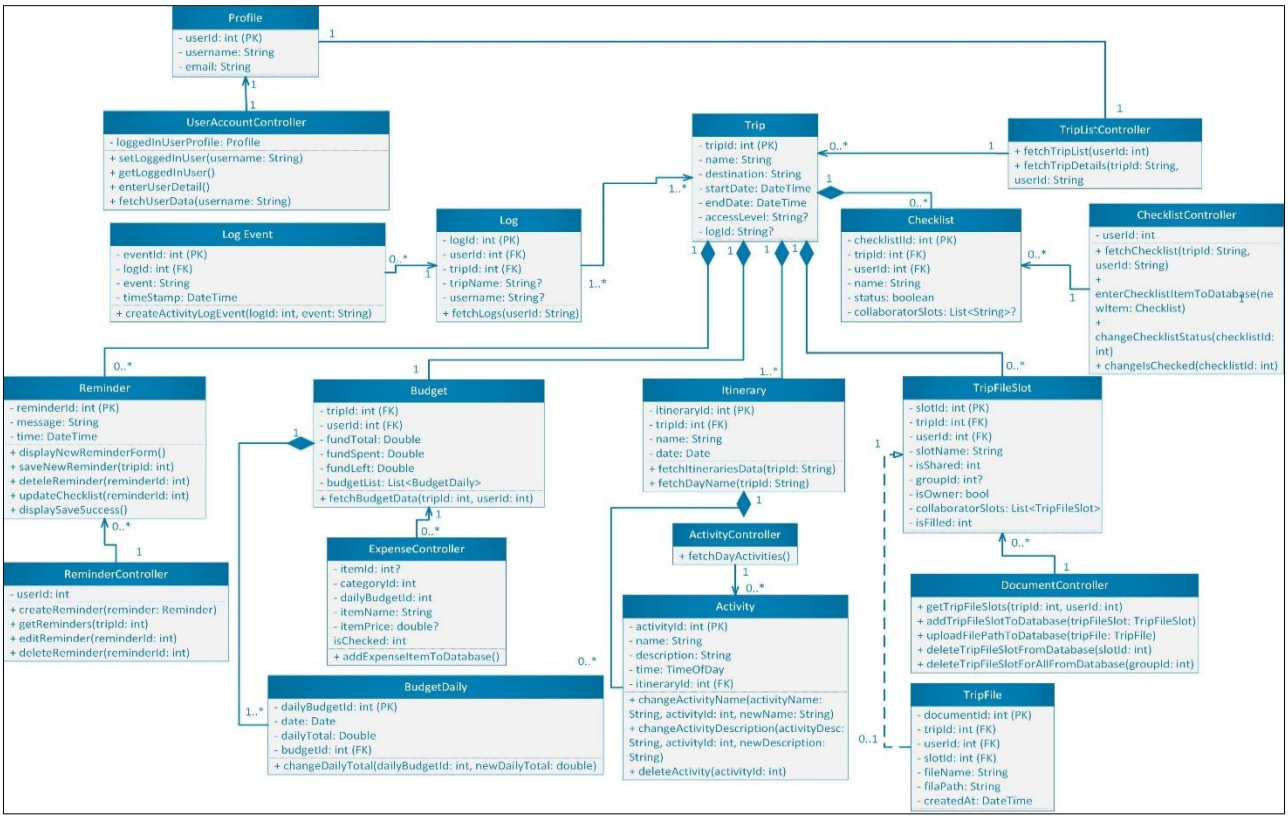


Fig. B2 *Class Diagram*

APPENDIX C: Deliverables during Design Phase

C.1 Data Models

Table C1 *TripController Data Dictionary*

No	Attribute	Data Type	Size	Key	Description
1	tripId	int	11	PK	Unique ID of trip
2	name	varchar	255	-	Name of trip
3	destination	varchar	255	-	Trip location
4	startDate	date	-	-	Start date
5	endDate	date	-	-	End date

Table C2 *Collaborations Data Dictionary*

No	Attribute	Data Type	Size	Key	Description
1	collaborationId	int	11	PK	Unique ID of collaboration
2	tripId	int	11	FK	ID of the trip this collaboration belongs to
3	userId	int	11	FK	ID of the user
4	accessLevel	varchar	50	-	Access level of user

Table C3 *Itinerary Data Dictionary*

No	Attribute	Data Type	Size	Key	Description
1	itineraryId	int	-	PK	Unique ID of itinerary
2	tripId	int	-	FK	ID of the trip this itinerary belongs to
3	name	varchar	255	-	Name of itinerary
4	date	Date	-	-	Itinerary date

C.2 User Interface Design

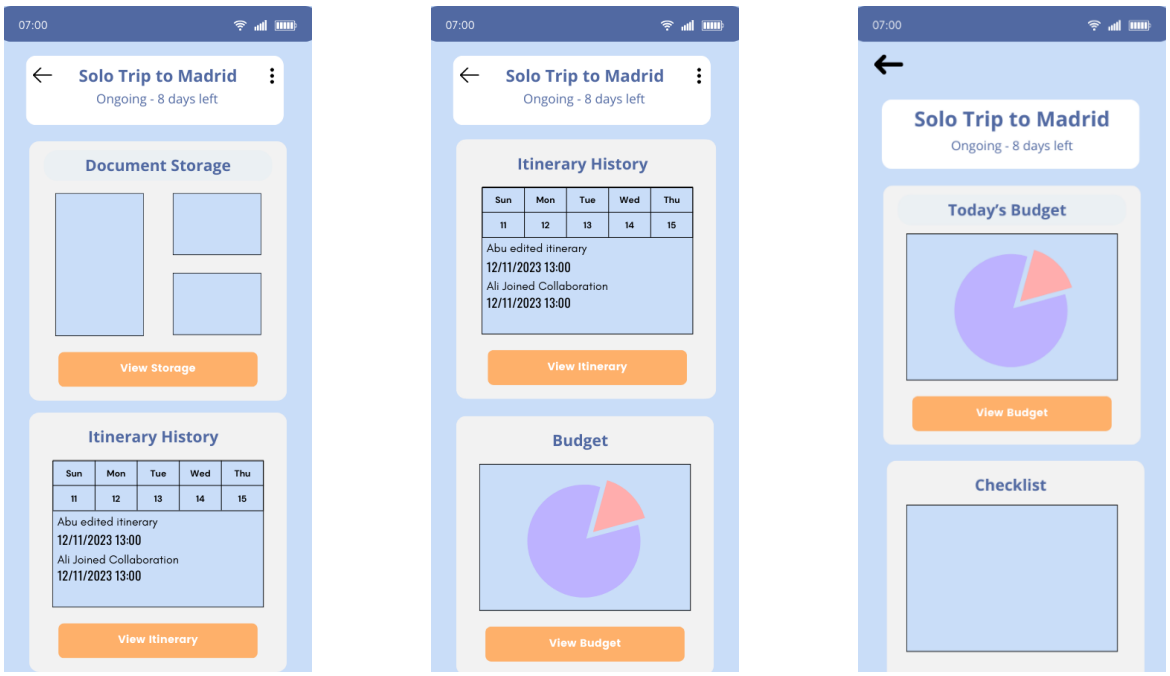
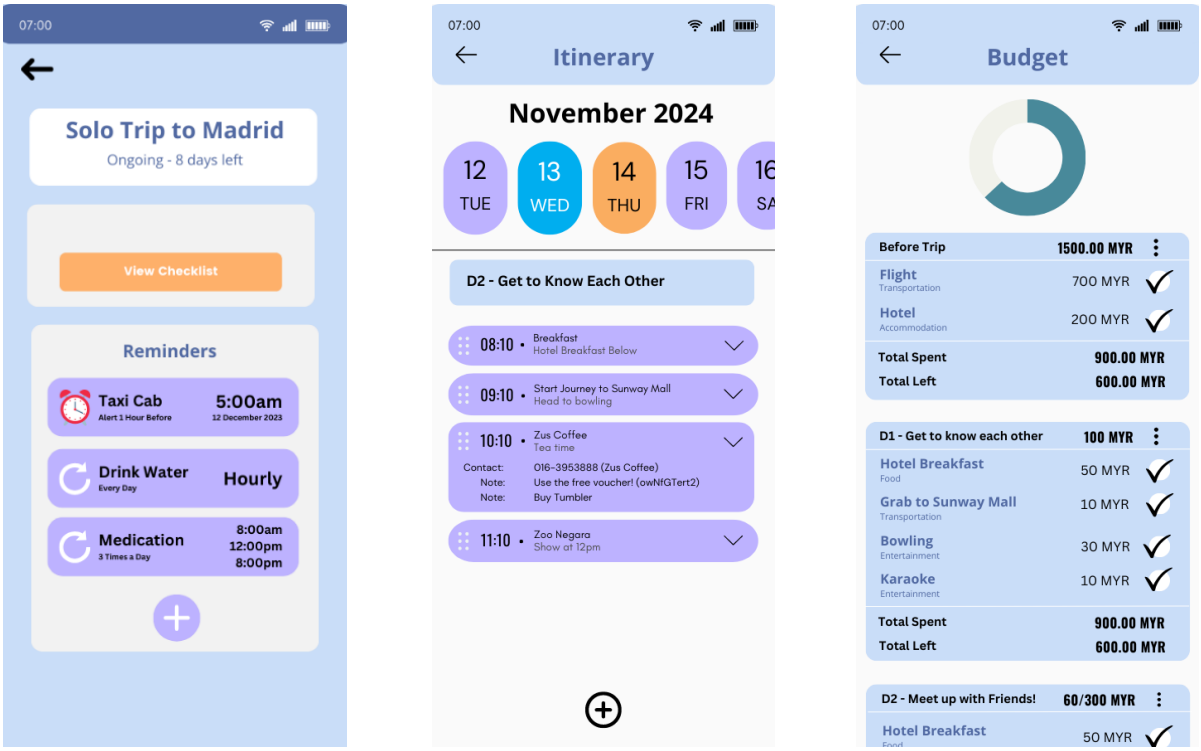


Fig. C1 *Modules of a Trip Plan (a) Document; (b) Itinerary; (c) Budget and Checklist*



(a) **(b)** **(c)**

Fig. C2 *Modules of a Trip Plan (a) Reminder module; (b) Itinerary; (c) Budget*

APPENDIX D: System Implementation Results

```
} else if (isset($_GET['createNewExpenseItem']) && $_GET['createNewExpenseItem'] === 'true') {
    if (
        isset($_POST['tripId']) && isset($_POST['dailyBudgetId']) && isset($_POST['itemName']) && isset($_POST['itemPrice']) &&
        isset($_POST['categoryId']) && isset($_POST['isChecked'])
    ) {
        $tripId = $_POST['tripId'];
        $dailyBudgetId = $_POST['dailyBudgetId'];
        $itemName = $_POST['itemName'];
        $itemPrice = $_POST['itemPrice'];
        $categoryId = $_POST['categoryId'];
        $isChecked = $_POST['isChecked'];

        // Prepare and execute the insert query using prepared statements
        $sql = "INSERT INTO expenseitem (categoryId, dailyBudgetId, itemName, itemPrice, isChecked)
                VALUES (?, ?, ?, ?, ?)";
        if ($stmt = $connectNow->prepare($sql)) {
            $stmt->bind_param("iisdi", $categoryId, $dailyBudgetId, $itemName, $itemPrice, $isChecked);
            if ($stmt->execute()) {
                $newItemId = $stmt->insert_id; // Retrieve last inserted ID using the insert_id property
                $response = array("success" => true, "newItemId" => $newItemId);

                // If selectedCollaborators is provided, find their dailyBudgetIds and create expense items
                $selectedCollaborators = isset($_POST['selectedCollaborators']) ? json_decode($_POST['selectedCollaborators']) : [];
                if (!empty($selectedCollaborators)) {
```

Fig. D1 *Logging New Expenses Code Segment*

```php
} else if (isset($_GET['addActivity']) && $_GET['addActivity'] === 'true') {
    if (isset($_POST['itineraryId'])) {
        $itineraryId = mysqli_real_escape_string($connectNow, $_POST['itineraryId']);

        $name = isset($_POST['name']) ? mysqli_real_escape_string($connectNow, $_POST['name']) : '';
        $description = isset($_POST['description']) ? mysqli_real_escape_string($connectNow, $_POST['description']) : '';
        $time = isset($_POST['time']) ? mysqli_real_escape_string($connectNow, $_POST['time']) : null;

        $sql = "INSERT INTO activity (itineraryId, name, description, time) VALUES ($itineraryId, '$name', '$description', '$time')";

        $resultOfQuery = $connectNow->query($sql);

        if ($resultOfQuery) {
            $newActivityId = $connectNow->insert_id;
            $newActivitySql = "SELECT * FROM activity WHERE activityId = $newActivityId";
            $newActivityResult = $connectNow->query($newActivitySql);
            $newActivityData = $newActivityResult->fetch_assoc();

            $activityDetailsSql = "INSERT INTO activitydetails (activityId, title, subject) VALUES ($newActivityId, NULL, NULL)";
            $activityDetailsResult
            = $connectNow->query($activityDetailsSql);

            if ($activityDetailsResult) {
                $response = array("success" => true, "data" => $newActivityData);
            } else {
                $response = array("success" => false, "error" => "Failed to add activity details");
            }
```

Fig. D2 *Add Activity Code Segment*

```php
// If collaborators are provided, create trip file slots for them
if (isset($_POST['collaborators'])) {
    // Get the first created slotId
    $firstSlotId = $slotId;

    // Retrieve collaborators
    $collaborators = explode(',', $_POST['collaborators']);
    foreach ($collaborators as $collaboratorId) {
        // Use the first slotId to make the groupId the same for all collaborators
        $groupId = $firstSlotId;

        // Insert trip file slots for collaborators with the same groupId
        $sql = "INSERT INTO tripfileslot (userId, tripId, slotName, isShared, groupId, isFilled) VALUES (?, ?, ?, ?, ?, 0)";
        $stmtInsertCollabSlot = $connectNow->prepare($sql);
        $stmtInsertCollabSlot->bind_param("iisii", $collaboratorId, $tripId, $slotName, $isShared, $groupId);

        if ($stmtInsertCollabSlot->execute() !== TRUE) {
            // Query execution failed
            $response = array(
                "success" => false,
                "error" => "Failed to create trip file slot for collaborator $collaboratorId: " . $connectNow->error
            );

            // Output the response as JSON
            header('Content-Type: application/json');
            echo json_encode($response);

            // Exit the script
            exit();
        }
    }
```

Fig. D3 *Creating New Trip File Slots Code Segment*

```php
} else if (isset($_GET['createNewChecklist']) && $_GET['createNewChecklist'] === 'true') {
    if (isset($_POST['tripId']) && isset($_POST['userId']) && isset($_POST['name']) && isset($_POST['status'])) {
        $tripId = $_POST['tripId'];
        $userId = $_POST['userId'];
        $name = $_POST['name'];
        $status = $_POST['status'];
        $collaboratorSlots = isset($_POST['collaboratorSlots']) ? json_decode($_POST['collaboratorSlots']) : [];

        $sql = "INSERT INTO checklist (tripId, userId, name, status) VALUES (?, ?, ?, ?)";
        $stmt = $connectNow->prepare($sql);
        $stmt->bind_param("iiss", $tripId, $userId, $name, $status);


        if ($stmt->execute()) {
            $newChecklistId = $stmt->insert_id;

            $response = array("success" => true, "newChecklistId" => $newChecklistId);

            foreach ($collaboratorSlots as $collaboratorId) {
                $collaboratorSql = "INSERT INTO checklist (tripId, userId, name, status) VALUES (?, ?, ?, ?)";
                $collaboratorStmt = $connectNow->prepare($collaboratorSql);
                $collaboratorStmt->bind_param("iiss", $tripId, $collaboratorId, $name, $status);
                $collaboratorStmt->execute();
                $collaboratorStmt->close();
            }
        } else {
            $response = array("success" => false, "error" => "Failed to add checklist item: " . $stmt->error);
        }
        $stmt->close();
        echo json_encode($response);
    } else {
        $response = array("success" => false, "error" => "Missing or invalid parameters");
        echo json_encode($response);
    }
```

Fig. D4 *Creating New Checklist Code Segment*

APPENDIX E: Functional Testing Results

Table E1 *Test Case Results for Budgets & Expenses Management Module*

No	Test Case Description	Expected Output	Actual Output	Status
1	Track budgets and expenses for a trip	Budgets and expenses are tracked correctly	As expected output	Passed
2	Update budget and expense details	Budget and expense details are updated successfully	As expected output	Passed
3	Delete expense records	Expense records are deleted successfully	As expected output	Passed
4	Change isChecked Flag	isChecked flag is updated successfully	As expected output	Passed
5	Change daily total	Daily total is updated successfully	As expected output	Passed
6	Edit expense item in database	Expense item is edited in database successfully	As expected output	Passed
7	Add expense item to database	Expense item is added to database successfully	As expected output	Passed
8	Owner add expense item for other collaborators to database	Expense item added by owner for collaborators is successfully recorded in the database	As expected output	Passed

Table E2 *Test Case Results for Trip Plan Management Module*

No	Test Case Description	Expected Output	Actual Output	Status
1	Create a new trip plan with all required details	Trip plan is created successfully	As expected output	Passed
2	View existing trip plans	Trip plans are displayed correctly	As expected output	Passed
3	Delete a trip plan	Trip plan is deleted successfully	As expected output	Passed
4	Attempt to create a trip plan with missing fields	System displays "All fields are required"	As expected output	Passed
5	Attempt to create a trip plan with invalid budget	System prevents invalid budget input	As expected output	Passed
6	Attempt to create a trip plan with past start date	System displays "Start date must be in the future"	As expected output	Passed
7	Attempt to create a trip plan with end date before start date	System displays "End date must be after start date"	As expected output	Passed
8	Attempt to create a trip plan without selecting collaborators for group trip	System displays "Please add collaborators"	As expected output	Passed
9	Successfully add collaborators to a group trip	Collaborators are added to the trip plan	As expected output	Passed
10	Attempt to add duplicate collaborators to a group trip	System prevents adding duplicate collaborators	As expected output	Passed
11	Successfully navigate back from creating a trip plan	User returns to previous screen	As expected output	Passed
12	Successfully create a solo trip plan	Solo trip plan is created	As expected output	Passed
13	Attempt to create a solo trip plan with collaborators	System prevents adding collaborators	As expected output	Passed

Table E3 *Test Case Results for Documents Management Module*

No	Test Case Description	Expected Output	Actual Output	Status
1	Upload a document for a trip	Document is uploaded successfully	As expected output	Passed
2	Monitor the status of document uploads	Document upload status is displayed correctly	As expected output	Passed
3	Delete a collaborator slot	Collaborator slot is deleted successfully	As expected output	Passed
4	Delete own slot	Own slot is deleted successfully	As expected output	Passed
5	Delete a trip file	Trip file is deleted successfully	As expected output	Passed
6	Delete a collaborator slot for all collaborators	Collaborator slot is deleted for all collaborators	As expected output	Passed

Table E4 *Test Case Results for Collaborators Management Module*

No	Test Case Description	Expected Output	Actual Output	Status
1	Invite a collaborator with a valid email	Collaborator invitation is sent successfully	As expected output	Passed
2	Remove a collaborator from a trip plan	Collaborator is removed successfully	As expected output	Passed
3	Update collaborator access level for a trip collaborator	Collaborator access level is updated successfully	As expected output	Passed

Applied Information Technology and Computer Science Vol. 5 No. 2 (2024) p. 480-499

Table E5 *Test Case Results for Trip Itinerary Management Module*

No	Test Case Description	Expected Output	Actual Output	Status
1	View existing trip itineraries	Trip itineraries are displayed correctly	As expected output	Passed
2	Update trip itinerary details	Trip itinerary details are updated successfully	As expected output	Passed
3	Add New Activity Details	New activity details are appended to the list	As expected output	Passed
4	Delete Activity Details	Deleted activity details are removed from the list	As expected output	Passed
5	Change Activity Title	Modified title of activity details is reflected	As expected output	Passed
6	Change Activity Subject	Updated subject of activity details is displayed	As expected output	Passed
7	Dismiss Activity Details	Dismissed details are removed, Last detail cannot be dismissed	As expected output	Passed
8	Change Activity Time	Activity time is updated successfully	As expected output	Passed
9	Change Activity Name	Activity name is updated successfully	As expected output	Passed
10	Change Activity Description	Activity description is updated successfully	As expected output	Passed
11	Delete Activity	Activity is deleted successfully	As expected output	Passed
12	Change Day Name	Day name is updated successfully	As expected output	Passed
13	Add Container Count	Activity count in the container is incremented	As expected output	Passed
14	Add Activity	New activity is added successfully	As expected output	Passed
15	Display Daily Budget and Expenses on the date	Daily budget and expenses is found for the specified date and displayed	As expected output	Passed
16	Reorder Activities	Activities are able to be reordered and is updated successfully	As expected output	Passed

APPENDIX F: User Acceptance Testing Functionalities Testing Results

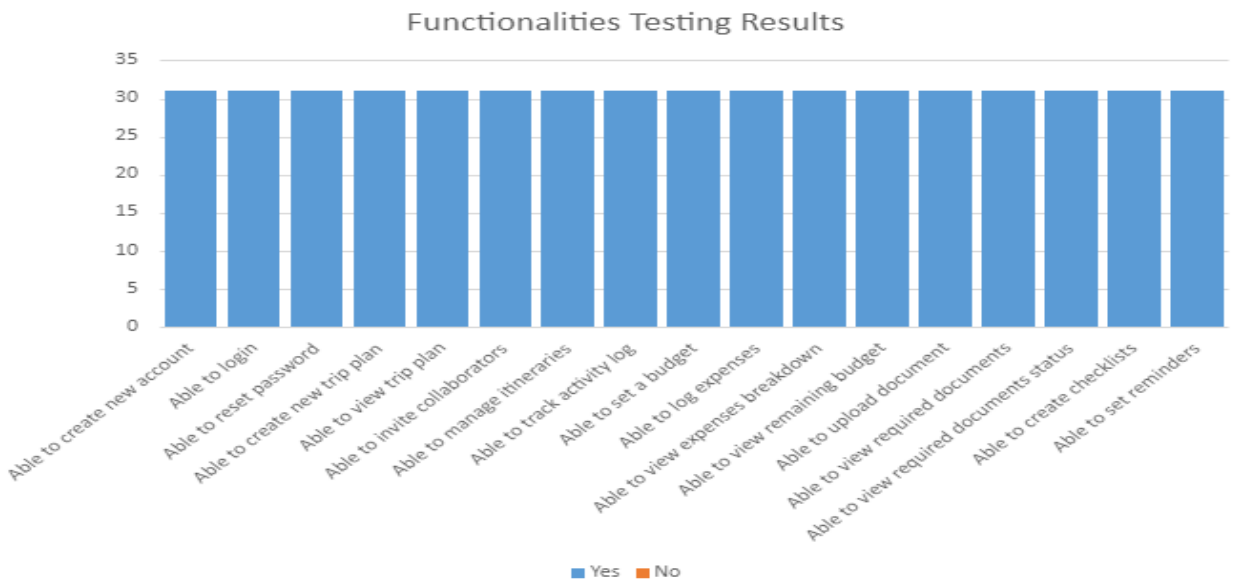


Fig. F1 *Functionalities Testing Results*