ExploreEasy: Smart and All-In-One Trip Management Application

BY

YAP PEI NEE

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

BACHELOR OF INFORMATION SYSTEMS (HONOURS) BUSINESS INFORMATION SYSTEMS

Faculty of Information and Communication Technology

(Kampar Campus)

JUNE 2025

COPYRIGHT STATEMENT

ACKNOWLEDGEMENTS

I would like to extend my heartfelt gratitude and appreciation to my supervisor, Ts.Yong Tien Fui, for granting me the invaluable opportunity to work on the EXPLOREEASY: SMART AND ALL-IN-ONE TRIP MANAGEMENT APPLICATION. His guidance and support have been invaluable throughout this journey. My heartfelt thanks for his encouragement.

To a very special person in my life, Jocelyn, thank you for your unwavering patience, love, and support. Your presence and encouragement during challenging times have meant the world to me.

Finally, I express my deepest gratitude to my parents and family for their endless love, support, and motivation throughout this journey. Their belief in me has been a constant source of strength.

ABSTRACT

As travellers increasingly seek tailored and efficient experiences, current travel applications often fail to address diverse requirements and adapt to real-time changes. This research presents an AI-driven trip planning and recommendation system that employs a Hybrid Recommendation Algorithm, integrating Collaborative Filtering (CF) and Content-Based Filtering (CBF) to provide highly customised travel itineraries. Core components include automated accommodation suggestions, an inflation-aware budget estimation and management module that applies Jaccard Similarity and Weighted Averaging for accurate budget ranges, and a cost split feature for fair expense sharing among travellers. The system also provides real-time budget alerts to enhance financial transparency and control. To improve travel efficiency, the system incorporates intelligent route optimisation using the Travelling Salesman Problem (TSP), ensuring time-efficient and logically sequenced itineraries. Additionally, a similar-place substitution feature leveraging Geographic Filtering and Quality Thresholds increases flexibility by dynamically suggesting contextually relevant alternatives. Furthermore, the integration of real-time and extended weather forecasting enables dynamic itinerary modifications to enhance safety and adaptability. The system's effectiveness was evaluated with real travel data, revealing significant improvements in personalisation, flexibility, financial confidence, and user satisfaction. By combining a hybrid recommendation engine with innovative features such as weather-aware itinerary adjustments, budget monitoring, expense splitting, and substitution-based adaptability, this project delivers a more intelligent, responsive, and user-centric trip planning experience than conventional platforms.

Keywords: AI-driven trip planning, Hybrid recommendation system, Budget tracking alerts, Route optimisation, Travelling Salesman Problem (TSP)

Area of study: Recommender Systems and Smart Tourism

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

$J(A,B)$	Jaccard similarity between sets A and B
$\sum$	Summation Operator
σ	Standard deviation
$\widehat{\mathrm{PPN}}$	Weighted average PPN estimate
β	Bandwidth parameter for budget range (0.05–0.15)
λ_j, λ_i	Longitudes of points i and j (in radians)
Δ	Difference operator
$\emptyset$	Latitude value

LIST OF ABBREVIATIONS

API	Application Programming Interface
AccTotal	Total Accommodation Cost
CBF	Content-Based Filtering
CF	Collaborative Filtering
CPP	Cost Per Person
ERD	Entity Relationship Diagram
IoU	Intersection over Union
NCF	Neural Collaborative Filtering
PPN	Per-Person-Per-Night
TSP	Traveling Salesman Problem
WA	Weighted Average

Chapter 1
Introduction

In this chapter, we present the problem statement, motivation, research objectives, project scope and direction, our contributions to the field, and the report organization.

1.1 Problem Statement and Motivation

This chapter provides an overview of the project by presenting the problem statement, motivations, objectives, scope, contributions, and the overall organization of the report.

1.1.1 Problem Statement

1. Many travellers struggle to plan cost-effective and satisfying trips due to financial limitations, overwhelming research requirements, and lack of destination familiarity, leading to suboptimal and stressful travel experiences.

Despite the abundance of travel-related applications already accessible, many travelers continue to encounter substantial obstacles in properly optimizing their travel experiences. A primary concern is that users have substantial difficulties in trip planning due to financial limitations, tedious research, and unfamiliarity with their chosen destinations, resulting in ineffective travel experiences and frustration in itinerary organization. Organizing a cost-effective trip frequently proves challenging, necessitating thorough study to evaluate rates for activities and lodging. In the absence of local expertise or dependable resources to identify economical choices, travelers must navigate extensive information, resulting in a process that is both time-consuming and stressful [1]. Budget-conscious tourists face increased challenges in making educated judgements when uncertain about obtaining optimal value for their expenditures. This complexity adds a significant amount of stress to the travel experience, as travelers are concerned that a poor choice could result in unsatisfactory experiences or unexpected expenditures, ultimately compromising the quality of their trip. As a result, travelers often encounter difficulties in locating options that align with their budgetary constraints and individual tastes, resulting in suboptimal selections that detract from the entire trip experience.

2. Group travelers frequently face confusion and conflict due to the lack of integrated tools for budget estimation, accurate shared expense tracking, equitable cost splitting, and real-time budget monitoring in current travel applications

Travelers often do not know how much to spend in specific areas because they have never visited those destinations before. Without prior knowledge or reliable reference data, they struggle to estimate realistic budgets, leading to either overspending or overly restrictive financial planning. Another significant concern encountered by group travelers is the difficulty in managing shared expenses accurately [2]. Managing shared expenses during group travel presents significant challenges, often leading to confusion, miscalculations, and interpersonal conflicts. Traditional methods, such as manual record-keeping or basic calculator applications, are prone to errors and lack essential features like real-time collaboration, expense categorization, and individual contribution summaries. Moreover, the absence of proactive budget monitoring tools in most travel applications results in unplanned overspending, with surveys showing that financial mismanagement contributes to conflicts among travel companions [3]. Therefore, there is a critical need for an integrated, automated budget management system that supports transparent expense tracking, equitable expense splitting, and real-time budget alerts to enhance financial coordination and improve the overall travel experience.

3. Most existing travel applications lack a reliable place-substitution feature, resulting in rigid itineraries that cannot adapt to unavailable or unsuitable locations, limiting flexibility, personalisation, and overall user satisfaction.

Most existing travel applications lack flexibility in itinerary planning, as they do not provide mechanisms for substituting places with contextually similar alternatives. When a recommended location is unavailable, unsuitable, or less preferred, users are forced to either remove it entirely or manually search for replacements, which disrupts the flow of their itinerary [4] .This limitation often leads to rigid travel plans that fail to adapt to users' evolving preferences, situational constraints, or quality concerns such as low ratings or overcrowding.

Without an intelligent substitution feature, travellers risk missing out on nearby options that are geographically feasible and equally relevant to their interests. The absence of such adaptability reduces personalisation and may result in dissatisfaction when planned activities do not align with expectations.

1.1.2 Motivation

The impetus for creating the ExploreEasy application arises from a comprehensive awareness of the complex obstacles encountered by contemporary travellers. A primary motivation is to provide a seamless and pleasurable travel experience for users, particularly for those who lack of time, finances, or knowledge to manage detailed trip planning. ExploreEasy is conceived as a solution that streamlines travel for anyone, from the experienced traveler to the infrequent holidaymaker. ExploreEasy seeks to decrease the stress and uncertainty frequently associated with travel by incorporating features like expense tracking and cost-effective trip planning. Additionally, a further boost for creating ExploreEasy is to address the inefficiencies and fragmentation inherent in existing trip management software. Numerous current applications provide restricted functionality and demand users to navigate multiple platforms for various parts of their journey, resulting in an unwieldy and fragmented experience. ExploreEasy aims to resolve this problem by offering a comprehensive solution that combines vital travel functionalities, including itinerary management, expense tracking, and booking. This integrated strategy seeks to streamline travel preparation, decrease user anxiety, and improve the overall efficacy of trip management, so rendering travel more attainable and pleasurable for all users. The motivation for this initiative stems from a dedication to inclusivity and accessibility. The travel industry has historically faced criticism for predominantly serving rich travellers, hence restricting possibilities for budget-conscious customers.[7]. ExploreEasy aims to equalise opportunities by providing tools that emphasise affordability while maintaining quality. This emphasis on inclusivity caters to travellers of diverse backgrounds and abilities, guaranteeing that all individuals, irrespective of their financial resources or travel experience, can navigate the world with assurance and simplicity.

1.2 Objectives

1. To develop an AI-driven trip planning and recommendation system using a hybrid recommendation algorithm with integrated real-time and extended weather forecasting for efficient, adaptive, and personalized travel planning.

The primary objective of this project is to create an AI-driven trip planning and recommendation system using a hybrid recommendation algorithm aimed at enhancing the travel planning process through increased efficiency and personalisation. This system will employ sophisticated AI to assess users' interests and financial limitations, producing personalised itineraries that include recommended places and lodging. Several essential sub-objectives have been identified to attain this goal. The project will provide a hybrid AI recommendation system that integrates **Collaborative Filtering (CF) and Content-Based Filtering (CBF)** to deliver highly personalised trip recommendations, even for individuals with less historical data. The technology will also offer **automatic accommodation recommendations** along the designated routes, hence increasing ease and optimising the trip planning process by minimising the necessity for users to utilize numerous applications. Additionally, the system will incorporate **intelligent route optimisation** using the **Travelling Salesman Problem (TSP)** approach, ensuring that the recommended itineraries are not only personalised but also time-efficient and logically sequenced to enhance the overall travel experience. Furthermore, the integration of **real-time and extended weather forecasting** into the itinerary planning module will enable dynamic adjustment of activities based on forecasted conditions, thereby reducing the risk of outdoor plans being disrupted by adverse weather, improving safety, enhancing flexibility, and increasing user satisfaction.

2. To implement a comprehensive budget estimation and management module using Jaccard Similarity and Weighted Averaging for accurate budget guidance and effective travel expense control.

The primary objective of this component is to design and integrate a budget system that leverages similarity-based techniques to provide travelers with reliable budget ranges when planning their trips, while also enabling effective management of expenses during travel. The estimation system analyzes historical trip data, computing per-person-per-day spending patterns from past users. Jaccard similarity is used to measure area overlap between trips, while weighted averaging accounts for traveler count and trip duration differences to ensure that the most relevant comparisons influence the estimate. To further improve accuracy, the system considers the inflation when generating budget estimation ranges, ensuring that cost predictions remain realistic and up to date. To enhance robustness, percentile-based outlier trimming is applied to exclude extreme spending values, and overspend filtering is used to remove trips where users exceeded their budgets by more than 10%. In cases of common trip profiles, exact matches in traveler count and day length are prioritized, yielding higher accuracy and narrower confidence ranges. Deterministic fallback rules (e.g., RM300 per person per day) ensure usability even with sparse data. In addition to estimation, the budget module supports allocation of travel budgets across categories such as accommodation, food, transport, and others, while also providing a cost split feature that allows expenses to be fairly divided among multiple travelers. Real-time alerts are provided to notify users when spending approaches or exceeds the allocated budget, enabling stronger financial awareness and control throughout the trip. By combining predictive budget estimation with proactive monitoring and expense splitting, this module enhances user confidence, decision-making, and financial discipline during travel.

3. To enhance flexibility in itinerary planning by implementing a reliable similar-place substitution feature using Geographic filtering and Quality thresholds.

The primary objective of this project is to provide travelers with the ability to dynamically substitute recommended places with contextually similar alternatives, thereby increasing personalization, adaptability, and reliability in trip planning. This feature ensures that users retain full control over their itineraries, while the system guarantees that alternate suggestions remain geographically relevant, reputable, and aligned with the user's preferences and conditions.Several essential sub-objectives have been identified to achieve this goal. The system incorporates **Geographic Filtering** using the Haversine formula to ensure suggested alternatives lie within a 2km radius of the primary location, maintaining feasibility within the daily route. **Quality Thresholds**, applied consistently during both dataset seeding and runtime queries, guarantee that only places with adequate ratings and reviews are offered as substitutes.

1.3 Project Scope and Direction

A key component of the proposed system is the budgeting feature, which provides users with comprehensive tools to manage and monitor their travel expenses. This module will include cost-splitting functionalities specifically designed for group travelers, allowing multiple users to collaboratively allocate and track shared costs with transparency and accuracy. Users will be able to assign expenses for accommodations, activities, meals, and other travel-related charges among group members, ensuring a fair and organized distribution of financial responsibilities.

A crucial component of the proposed system is the planning feature, which offers customers AI-driven tools to efficiently organise and enhance their travel schedules. This feature will include an advanced trip planning system that employs artificial intelligence to create tailored itineraries based on user preferences, destination, and budget.

Finally, the elements of the proposed system include an accommodation booking feature. Users can directly reserve lodgings through the app. By providing an integrated platform for securing accommodation, the system aims to streamline the planning process and enhance user convenience. This feature ensures that travelers can efficiently organize their stay within the same platform used for itinerary management, contributing to a more cohesive and user-centered travel experience.

1.4 Contributions

1. AI planning automates routes and suggestions for a faster, all-in-one travel experience. (Addressing problem#1)

The AI-driven trip planning and recommendation system markedly improves the travel planning experience by offering consolidated hotel recommendations and budget- oriented route generation, thereby obviating the necessity for users to depend on several applications. This holistic approach guarantees that all facets of the journey are addressed together, resulting in a more fluid and integrated experience. Furthermore, the system conserves users' time by automating the arrangement of ideal routes, a process that is conventionally laborious and intricate. By automating and optimising these procedures, the system enhances decision-making and expedites the whole planning procedure. In conclusion, the system surpasses existing alternatives in both speed and quality, providing users with a more streamlined, efficient, and gratifying method for trip planning.

2. Budget tracking is automated for clear, fair, and stress-free expense management (Addressing problem#2)

The integrated budget module enhances the travel experience by combining budget estimation and expense management into a single, automated feature. Before a trip, the system provides travellers with realistic budget ranges tailored to their destination, trip duration, and group size, helping them plan with confidence. During the trip, it automates expense tracking and eliminates the need for manual calculations, which are often error-prone and tedious. It offers users a comprehensive overview of both individual and shared expenses, improving transparency and promoting equitable financial coordination among group travellers. The system enables users to add, edit, categorise, and split expenses seamlessly within the application, while also supporting the upload of receipts for better accountability. Additionally, real-time budget monitoring with automated alerts notifies users when they are nearing their spending limits, allowing them to adjust their plans proactively. This holistic approach ensures that travellers not only start with reliable budget guidance but also maintain ongoing financial control, resulting in a more streamlined, accurate, and stress-free travel experience.

3. Flexible substitution enhances adaptability and user satisfaction (Addressing problem#3)

The itinerary module enhances the travel experience by giving users the flexibility to substitute recommended places with suitable alternatives when their first choice is unavailable, overcrowded, or less appealing. Instead of forcing travellers to manually search for replacements, the system automatically suggests nearby and high-quality options that fit seamlessly into their existing route. This ensures that plans remain practical, enjoyable, and aligned with individual preferences, even when unexpected situations arise. By maintaining itinerary coherence while still offering choice, the feature empowers travellers with greater control, reduces the stress of last-minute adjustments, and creates a more personalised and resilient travel experience.

1.5 Report Organization

The details of this research are organized into the following chapters. Chapter 2 reviews algorithm used and existing travel planning and recommendation systems, highlighting their features and limitations. Chapter 3 presents the system design, including UML diagrams, the methodology adopted, the project timeline, the development tools utilized, discussion of the implementation issues and challenges encountered, and implemented algorithms and technologies. Chapter 4 showcases Blackbox testing, Client Satisfaction Survey Analysis, Results and Benchmark ,Objectives Evaluation and Concluding Remark . Finally, Chapter 5 provides the conclusion, summarizing the project objectives achieved.

Chapter 2
Literature Review

2.1 Algorithm

2.1.1 Collaborative Filtering (CF)

Collaborative Filtering (CF) is a core technique in recommender systems that generates personalized suggestions by exploiting the collective behaviors and preferences of users. It operates on the principle that individuals with similar past interactions are likely to share future interests, typically by analyzing user–item interaction data such as ratings, clicks, or purchases [5].

2.1.1.1 Strengths

A key strength of CF is that it does not require explicit knowledge of item attributes, making it broadly applicable across different domains. It is capable of uncovering hidden patterns in user behavior and providing serendipitous recommendations, often introducing users to items they might not have actively searched for but are valued by similar peers. This "social proof" aspect enhances user trust and engagement, while advanced model-based approaches such as matrix factorization improve scalability and prediction accuracy by addressing sparsity in large datasets [6], [7]. More recent developments like Neural Collaborative Filtering leverage deep learning to capture complex non-linear relationships in user–item interactions, further boosting recommendation quality [8].

2.1.1.2 Weaknesses

Nonetheless, CF faces several limitations. Memory-based methods are computationally intensive and struggle to scale efficiently in large systems. They also suffer from the cold start problem, where insufficient data for new users or items prevents reliable recommendations, and data sparsity, where limited user interactions reduce the effectiveness of similarity calculations [5], [6]. Even though model-based and neural methods mitigate some of these challenges, they require significant computational resources and large datasets, making them less practical for smaller-scale applications [7], [8].

2.1.2 Content-Based Filtering (CBF)

Content-Based Filtering (CBF) is a recommendation technique that generates suggestions by analyzing the attributes of items and aligning them with a user's established preferences, rather than relying on the behavior of other users. The approach involves constructing a user profile from the features of previously interacted items—such as genres, keywords, or authors—and recommending new items that share similar characteristics. This makes CBF particularly valuable in scenarios where collaborative data is limited, as it can still provide recommendations without requiring overlapping preferences among multiple users. A common example is in news recommendation systems, where the topics and keywords of articles a user has read are leveraged to suggest other articles with related themes [9].

2.1.2.1 Strengths

The strengths of CBF lie in its ability to address the new item cold start problem and its provision of highly personalized recommendations. Since the system depends on item metadata rather than user ratings, it can immediately recommend new content once descriptive attributes are available, regardless of whether other users have interacted with it. This independence from collective user data allows the system to tailor suggestions closely to an individual's tastes while also offering potential privacy advantages, as the recommendations are derived solely from the user's own profile [10].

2.1.2.2 Weaknesses

However, CBF also has significant drawbacks. One major limitation is the reduced diversity of recommendations, as the system tends to repeatedly suggest items similar to those already consumed, which can create a "filter bubble" and limit opportunities for serendipitous discovery. Additionally, the accuracy of recommendations is highly dependent on the quality and richness of item metadata; if the descriptive features are incomplete or poorly structured, the system's ability to construct meaningful profiles and generate relevant suggestions is weakened [10].

2.1.3 Jaccard Similarity

Jaccard similarity, also referred to as the Jaccard index or Intersection over Union (IoU), is a statistical measure used to evaluate the similarity between two sets. It is calculated as the ratio of the intersection to the union of the sets, expressed mathematically as

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

. The coefficient ranges from 0 to 1, where a value of 1 indicates complete similarity and 0 indicates no overlap [11]. Owing to its simplicity and intuitive interpretation, the Jaccard index has been widely applied in diverse fields such as gene sequence analysis in bioinformatics, document comparison in text mining, and object detection in image processing.

2.1.3.1 Strengths

A key strength of the Jaccard similarity measure is its ease of interpretation and suitability for sparse data environments. Unlike frequency-based metrics, it relies solely on the presence or absence of elements, making it particularly useful for applications such as analyzing user preferences through shared items or evaluating common tags across web pages [12]. The straightforward computation, bounded range, and emphasis on shared elements rather than dataset size provide a clear and reliable means of assessing set overlap.

2.1.3.2 Weaknesses

Nevertheless, the measure is not without drawbacks. One limitation is its sensitivity to disparities in set size, which can distort results when comparing significantly imbalanced datasets. Additionally, it disregards the frequency of elements, treating single occurrences the same as multiple ones, thereby limiting its effectiveness in contexts where frequency carries meaningful weight compared to alternatives such as cosine similarity. Another issue is the "empty set problem," where the metric becomes undefined if both sets contain no elements [13]. These challenges highlight the need for careful dataset evaluation before employing the Jaccard coefficient.

2.1.4 Weighted Averaging

Weighted averaging expands on the basic arithmetic mean by allowing different levels of importance to be assigned to individual data points. The weighted average of a set of values $\{x_1, x_2, \ldots, x_n\}$ with associated weights $\{w_1, w_2, \ldots, w_n\}$ is calculated as:

$$WA = (\sum_{i=1}^{n} w_i x_i) / (\sum_{i=1}^{n} w_i)$$

2.1.4.1 Strengths

The strength of weighted averaging lies in its ability to normalize contributions, integrate multiple factors, and maintain proportional influence across components.The denominator ensures the weights are normalized. This method enables the integration of multiple factors while maintaining the proportional influence of each component [14].By assigning suitable weights, each data point contributes according to its significance, resulting in more accurate and meaningful outcomes—especially when factors such as cost, convenience, or user ratings vary in importance during trip planning [14].Additionally, down-weighting extreme or less reliable values minimizes the effect of outliers, producing more robust recommendations and enhancing the reliability of aggregated outputs [15]. Furthermore, Weighted Averaging is flexible and applicable across contexts, from expert-based risk evaluation in tourism to multi-criteria decision-making in travel planning. Beyond tourism, it is widely used in domains such as finance, engineering, and analytics due to its adaptability [16].

2.1.4.2 Weaknesses

However, assigning weights can introduce subjectivity, as they may be determined through expert judgment, user feedback, or algorithmic estimation. Poorly justified weights risk embedding bias into the results, thereby reducing their reliability [17]. Furthermore, even minor adjustments to weights or input data can significantly alter the aggregated output, leading to unstable or hard-to-interpret results, particularly in dynamic settings where preferences and data quality vary over time [15].

2.1.5 2-Opt Heuristic for the Traveling Salesman Problem

The Traveling Salesman Problem (TSP) represents one of the most fundamental combinatorial optimization problems in computer science, seeking to find the shortest possible route that visits each city exactly once and returns to the origin city. Due to its NP-hard nature, exact algorithms become computationally intractable for large instances, necessitating the development of efficient heuristic approaches. Among these heuristics, the 2-Opt algorithm stands out as one of the most widely adopted local search techniques for TSP optimization, offering an excellent balance between solution quality and computational efficiency.

The 2-Opt heuristic is a classic local-search method for the Euclidean Traveling Salesman Problem (TSP): starting from any tour, it repeatedly replaces two edges replace $(u,v),(w,z)$ with $(u,w),(v,z)$ whenever this shortens the tour, and stops at a *2-optimal* tour where no such improving swap exists. Equivalently, a tour T is 2-optimal if for every two edges appearing in order $(u,v),(w,z)$ on T it holds that $d(u,v)+d(w,z) \leq d(u,w)+d(v,z)$(otherwise a 2-opt move improves the tour). Künnemann and Manthey formalize the quantities $TSP(X)$ (length of an optimal tour) and $2OPT(X)$ (length of the *worst* 2-optimal tour) for a point set $X \subset [0,1]^d$, and study the *smoothed* approximation behavior of 2-Opt when adversarial instances are perturbed by small Gaussian noise with standard deviation σ [18].

2.1.5.1 Strengths

Although 2-Opt has weak worst-case guarantees, the paper explains part of its good empirical behavior via smoothed analysis: they prove an *upper bound* of $e^{O(\log(1/\sigma))}$
on the smoothed approximation ratio $2OPT(X)/TSP(X)$. This bound is strictly better than the $O(log\ n)$ worst-case guarantee when the perturbation is not vanishingly small (e.g., when $1/\sigma \leq (log\ n)^c$ for some constant ccc). A key technical novelty is that, rather than bounding the global optimum and a local optimum on different instances, they simultaneously bound both on the *same* perturbed instance—yielding a more realistic explanation of why modest measurement noise can eliminate pathological configurations. They also note that the insights transfer to a one-step perturbation model (bounded densities), underscoring that the conclusions are not tied to a single noise model [18].

2.1.5.2 Weaknesses

The authors emphasize that 2-Opt's worst-case behavior remains poor: its approximation ratio is $O(lo\,g\,n)$ with an almost matching lower bound $\Omega(lo\,g\,n\,/lo\,g\,l\,o\,g\,n)$, and even its worst-case running time can be exponential (already in $d=2$). Moreover, the lower-bound gap is robust to small noise: they show the $\Omega(lo\,g\,n\,/lo\,g\,l\,o\,g\,n)$ phenomenon persists under Gaussian perturbations of size $\sigma = O(1/\sqrt{n})$. They further caution that their analysis bounds the *worst* local optimum, which need not be the one reached in practice, and that real systems typically start from constructive heuristics (e.g., nearest-neighbor or insertion); indeed, even with a nearest-neighbor initialization, 2-Opt can still return a logarithmically worse 2-optimal tour with constant probability on perturbed inputs. Analyzing the smoothed approximation with realistic initializations is left as an open problem, so the theory does not fully explain all empirical successes [18].

2.2 Existing Travel Management System

2.2.1 Budget and Experience Based Travel Planner

The "Budget and Experience Based Travel Planner," proposed by Suwarno and Maziya Azelicha Ayana[19], is an advanced tool intended to assist travellers in organising vacations that match to their financial limitations while maintaining the quality of their experiences. This system utilises a combination of sophisticated routing algorithms, collaborative filtering, and web scraping to suggest itineraries that achieve optimal cost and user experience. This technology resolves the issue faced by budget-conscious users in identifying the most economical travel routes by leveraging data aggregated from multiple web sources. The system extracts data from travel-related websites utilising libraries such as BeautifulSoup and Selenium, collecting information including location coordinates, ratings, reviews, and displayed visit durations. This data supports the recommendation engine, which utilises collaborative filtering to propose locations that correspond with the user's interests and financial constraints. The system includes a routing algorithm that optimises the sequence of visited locations, minimising the overall distance travelled and thereby lowering travel expenses. Furthermore, the scheduler algorithm guarantees that the travel itinerary is compatible with the user's budget and time constraints by dynamically modifying it as required.

2.2.1.1 Strengths

The Budget and Experience Based travel Planner excels in delivering tailored and economical vacation suggestions, especially for those mindful of their expenses. The system employs web scraping to collect comprehensive data from diverse internet sources, enabling the customisation of travel plans according to individual preferences, budgetary limitations, and time restrictions [20]. This data-driven personalisation guarantees that travellers obtain up-to-date recommendations that correspond with their specific requirements. The system's incorporation of Collaborative filtering and the Haversine formula enables it to suggest both the most relevant destinations and the most economical routes connecting them [21]. This method effectively integrates budgetary factors into travel planning, guaranteeing that users receive the most economical choices without sacrificing the quality of their experience. Furthermore, this system incorporates an intelligent budget allocation mechanism that dynamically distributes the user-defined travel budget across key categories such as accommodation, food, transportation, and activities. This feature ensures that all recommended places and itinerary components remain financially feasible, reducing the risk of overspending.

2.2.1.2 Weaknesses

Nonetheless, despite these advantages, the system has certain obstacles and restrictions. A significant restriction is the reliance on web scraping to collect data from Google Travel and Google Maps. This approach has the potential to produce valuable information; however, it is susceptible to inconsistencies because of modifications to website structures or restrictions on scanning. This poses the risk of acquiring incomplete or outdated data, thus compromising the dependability of the suggestions. Additionally, the collaborative filtering technique, despite its effectiveness, also poses obstacles. It experiences the cold-start problem, wherein new users with minimal data obtain imprecise recommendations. Moreover, collaborative filtering depends on extensive datasets to provide significant outcomes, posing a problem when the sample is limited or incomplete [19]. Another key limitation is the absence of a budget estimation range, which leaves users uncertain about realistic spending expectations and increases the likelihood of overspending or underplanning. Similarly, the system lacks flexibility in selecting alternative choices; once an activity or location is unavailable or unsuitable, users are forced to manually search for replacements, often disrupting the itinerary flow. Furthermore, the routing algorithm, while proficient at reducing trip distances, exhibits a degree of inflexibility. It concentrates on distance at the expense of other critical factors, including traffic, transportation modes, and user preferences. In real-world scenarios where conditions can alter dynamically, this simplistic approach may not provide optimal routes.

2.2.1.3 Solution to solve weaknesses

There are various ways to improve the Budget and Experience Based Travel Planner in order to overcome its constraints. To mitigate dependence on web scraping for data collection, using APIs as a replacement or supplement would enhance data reliability. APIs from reputable travel platforms such as Google Places or TripAdvisor would deliver current, organised data on destinations, reviews, and ratings [22]. This would guarantee the system's accuracy and avoid the constraints of online scraping, including outdated information or modifications in website architecture.

Furthermore, to mitigate the shortcomings of collaborative filtering, a hybrid recommendation system integrating both collaborative filtering and content-based filtering may be employed [23]. Content-based filtering utilises destination qualities such as tags, ratings, and reviews to recommend locations, especially for new users with inadequate history data. This would help reduce the cold-start issue by providing personalised recommendations despite insufficient user data.

In addition, to address the absence of a budget estimation range, an integrated budget estimation module can be introduced. By analysing historical spending patterns, destination costs, and inflation adjustments, the system can provide users with realistic budget ranges before travelling. This would enable travellers to plan more effectively, avoid overspending, and improve financial confidence during their trips.

To overcome the lack of flexibility in selecting alternative choices, a similar-place substitution feature can be implemented. This feature would leverage geographic filtering to suggest feasible alternatives within a nearby radius and apply quality thresholds such as ratings and reviews to maintain recommendation standards. This ensures that itineraries remain adaptable and user-friendly even when original locations are unavailable, unsuitable, or less appealing.

Finally, enhancing the routing algorithm to incorporate real-time variables such as traffic conditions, modes of transportation, and user preferences would render the system more flexible. This would provide more effective route planning that prioritises user convenience and travel satisfaction, rather than merely minimising distance.

2.2.2 Wanderlog

Wanderlog, introduced in 2019, is an all-encompassing holiday planning application that prioritises user-friendly features. It enables users to plan and track itineraries, investigate places, and effectively oversee travel expenditures. The Wanderlog AI Trip Planning Feature tackles the difficulties identified in the problem statement. It streamlines the time-consuming process of trip planning by producing automated itineraries according to user preferences, assisting travellers in navigating the challenges of organising a trip, particularly when unfamiliar with the area they are visiting [24]. The AI offers customised suggestions for destinations, enabling customers to effectively assess alternatives without in-depth research. By taking user budgets into account, the AI mitigates apprehensions regarding the discovery of economical alternatives, enabling travellers to pinpoint affordable activities and lodgings that align with their financial constraints. Furthermore, the AI's capacity to enhance travel routes and modify recommendations according to user feedback alleviates the complexity and frustration frequently linked to trip planning. The AI provides a more efficient, personalised solution for travellers, directly tackling budget limitations, extensive research requirements, and unfamiliarity with the specified places. Besides its planning functionalities, Wanderlog tackles the prevalent issue of handling shared spending during group excursions. The application has an expense tracking feature that allows users to record and categorise all expenditures, providing transparency during the journey [25]. Furthermore, real-time collaboration enables all group members to rapidly edit and view shared spending, while the multi-currency conversion tool facilitates the tracking of costs across many currencies for foreign travellers. Collectively, these attributes establish Wanderlog as a comprehensive solution for vacation planning and financial management.

2.2.2.1 Strengths

A notable strength of Wanderlog's AI trip planner is the incorporation of an AI Travel Assistant powered by ChatGPT, which markedly improves the trip planning experience [26]. This chatbot facilitates an interactive planning process, enabling users to provide real-time feedback and receive prompt modifications to their trip routes and itineraries. This degree of involvement provides a more tailored and adaptable method, enabling users to customise their plans based on their preferences, including modifying travel times, changing destinations, or incorporating certain activities. The AI chatbot guarantees that the itinerary is customised to the user's requirements and is flexible to accommodate any modifications or new concepts that may emerge throughout the planning phase. This dynamic feature distinguishes Wanderlog by enhancing the trip planning process to be more user-centric and adaptive, hence facilitating a more fulfilling and pleasurable travel experience for users. Moreover, a principal advantage of the comprehensive expense tracking solution is its cost splitting functionality, which facilitates seamless division of expenses among group members, thereby obviating manual calculations and minimising confusion regarding financial obligations. Furthermore, Wanderlog's real-time collaboration feature enhances its utility for group travel by allowing all participants to promptly update and access shared charges. This guarantees that all parties are aware of current expenses, promoting transparency and reducing any misinterpretations. The multi- currency conversion option is especially beneficial for international travellers, as it facilitates the monitoring of expenses in several currencies, negating the necessity for external tools. The thoughtfully crafted features establish Wanderlog as an effective and intuitive tool for travel organisation and financial oversight.

2.2.2.2 Weaknesses

Nonetheless, the Wanderlog AI trip planner possesses notable deficiencies that could negatively affect the planning experience for travellers. A significant disadvantage is the absence of automated accommodation recommendations during route planning. The AI efficiently generates and modifies travel itineraries but does not offer hotel alternatives along those routes. This limitation requires users to actively search for and incorporate accommodations themselves, which can be laborious and inconvenient, particularly for individuals unfamiliar with the destination.

Furthermore, the AI lacks integration of weather forecasting, an essential component of trip planning, especially for outdoor activities. In the absence of integrated weather forecasts, users are compelled to depend on external sources to check weather conditions, hence complicating the planning process. This constraint can be especially exasperating for travellers who must manually modify their itinerary due to weather fluctuations.

Another weakness is the absence of budget estimation and allocation features. The system does not provide users with an initial budget range for their destination, nor does it allow allocation of budgets across categories such as accommodation, food, and transportation. Without these functions, travellers risk overspending or underestimating costs, reducing their confidence in financial planning. Additionally, there are no spending alerts to notify users when they approach or exceed their budget thresholds.

Lastly, Wanderlog does not offer a similar-place substitution feature. When recommended places are unavailable, overcrowded, or less appealing, users must manually search for alternatives. This lack of flexibility can result in rigid itineraries that fail to adapt to real-world conditions, reducing personalisation and overall satisfaction.

These deficiencies highlight opportunities for enhancing the AI trip planner into a more comprehensive, adaptive, and user-centric system that addresses accommodation, budgeting, weather-aware planning, and substitution flexibility.

2.2.2.3 Solution to solve weaknesses

To rectify the absence of automatic hotel recommendations, the AI trip planner can be improved by incorporating a feature that suggests lodging options along the designated route. By providing recommendations based on criteria such as budget, location, and amenities, users would enjoy a more streamlined experience, enabling them to organise their entire trip, including accommodation, within the same platform.

The lack of weather forecasting integration can be addressed by embedding a weather-aware feature directly into the trip planning process. By analysing forecasted conditions, the planner could automatically arrange outdoor activities on suitable days and suggest indoor alternatives when adverse weather is expected. This would give users greater confidence and minimise disruptions caused by unexpected conditions.

To overcome the absence of budget allocation and alert functionalities, the system can be enhanced by allowing users to set budget limits across categories such as accommodation, food, and transport. Real-time notifications could then alert users when expenses approach or exceed these limits, helping them maintain financial control and reduce the risk of overspending.

In addition, the lack of a budget estimation feature can be solved by providing users with an estimated spending range for their destination before travel. This would help travellers plan more effectively, set realistic expectations, and improve financial confidence.

Finally, the absence of a substitution feature can be addressed by introducing a similar-place replacement option. When a recommended location is unavailable, overcrowded, or less appealing, the planner could suggest nearby alternatives of comparable quality, ensuring itineraries remain flexible, practical, and satisfying for the user.

2.2.3 TripAdvisor

TripAdvisor is a platform that provides several services in the travel industry, mostly focused on user-generated ratings and suggestions. Users engage on the platform by sharing their experiences, offering ratings, and commenting on various locations or tourist attractions globally. Established in February 2000 as a travel search engine, TripAdvisor has evolved into one of the largest online travel platforms, significantly influencing the travel industry [27]. TripAdvisor's developers have addressed the obstacles encountered by budget-conscious travellers and those unfamiliar with their destinations by incorporating AI-driven trip planning via the "Trips" function. This solution assists travellers in navigating the daunting task of planning an economical trip by offering tailored suggestions for accommodations, dining, and activities [28]. The AI leverages user preferences, budgetary constraints, and historical behaviour to propose customised choices, thereby considerably minimising the necessity for exhaustive research. The AI sifts through extensive data, encompassing over 8 million locations and more than one billion traveler ratings, to deliver the most pertinent and economical options. Furthermore, users can preserve their selected preferences on a personalised map, facilitating enhanced visualisation and organisation of their journey. By automating the recommendation process, the AI assists users in making educated selections, hence mitigating the risk of poor choices and unforeseen expenses. This tailored method not only conserves time but also guarantees that travellers adhere to their budget while enhancing their overall experience. TripAdvisor has incorporated many transportation-related functionalities to assist consumers in planning their journeys. The platform enables users to search for flights and vehicle rentals while comparing prices from many booking providers. TripAdvisor consolidates transit options from several platforms, enabling travellers to view an extensive array of possibilities [29]. The application evaluates several airlines, fare classes, and durations, enabling users to discover bargains and make informed choices regarding trips. Moreover, TripAdvisor features evaluations and recommendations from fellow travellers, which may occasionally provide insights into economical transit options [29]. Nevertheless, although the platform provides a thorough summary of available services, it is deficient in functionalities for controlling group travel expenses and lacks detailed, localised suggestions for specialised, budget-friendly experiences. Notwithstanding these constraints, the platform's consolidation of services, together with user-generated content, continues to be a valuable resource for travellers seeking assistance with many facets of their trip planning.

2.2.3.1 Strengths

TripAdvisor's AI-driven trip planning solution possesses numerous significant advantages in tackling the difficulties faced by budget-conscious travellers and individuals unfamiliar with their destinations. The personalised recommendations feature customises choices according on user preferences, budget, and historical behaviour, facilitating the process for travellers to discover pertinent possibilities without extensive study. This directly tackles the problem of excessive information by sifting through extensive data to offer only economical options that align with the user's requirements. Moreover, the capacity to depict these stored locations on a personalised map significantly improves the planning process, enabling users to systematically arrange routes and supplies. The capability to exchange and agree on travel itineraries streamlines group travel, minimising planning difficulties and assuring consensus on essential decisions.

2.2.3.2 Weakness

Nonetheless, there are specific limitations to the proposed solution. The inability of the artificial intelligence to incorporate real-time weather predictions in planning trips is a major limitation. Because of this shortcoming, tourists can inadvertently schedule outdoor activities on days with unfavorable weather conditions, like excessive rainfall or extreme temperatures, and thus incur delays and compromise the quality of the trip as a whole. If weather conditions are not taken into consideration, the artificial intelligence may suggest destinations or travel plans that are not possible during certain seasons, thereby lowering the reliability and flexibility of the suggestions.

TripAdvisor's primary deficiency in transportation planning for budget-conscious travellers is its inadequate support for multimodal transit alternatives. Multimodal transportation denotes the integration of various transport modalities—such as aircraft, buses, trains, ferries, and ride-sharing services—to formulate a more economical or comfortable travel itinerary. This flexibility is essential for budget-conscious consumers, enabling them to discover more economical routes by utilising many forms of transportation instead of depending exclusively on a single way, such as direct flights.

Another notable weakness is the absence of budget estimation and allocation features. TripAdvisor does not provide users with an estimated spending range for destinations, nor does it allow them to assign budgets across categories such as accommodation, food, and transport. Without these tools, travellers may overspend, underestimate costs, or struggle to manage their finances effectively during a trip.

Finally, the system lacks a similar-place substitution feature, meaning users cannot easily replace unavailable, overcrowded, or less suitable locations with relevant alternatives. This limitation reduces itinerary flexibility and personalisation, as travellers must manually search for replacements, often disrupting the overall flow of their plans.

2.2.3.3 Solution to solve weaknesses

The website can integrate seasonal trends and real-time weather information into the recommendation system to counter the problem of trip planners failing to account for weather. By incorporating precise forecasting and climatic information, TripAdvisor can help consumers make more informed decisions regarding the ideal time to visit particular destinations or organise specific activities. Allowing tourists to avoid delays caused by bad weather and making their schedules both productive and enjoyable would also enhance the entire planning process.

To address TripAdvisor's deficiency in accommodating multimodal transit alternatives, the platform may include a comparison tool that assesses routes based on transfer frequency and walking lengths. This system could autonomously propose combinations of transport modalities customised to customers' preferences, such as utilising trains followed by local buses or bike-sharing services. By incorporating these alternatives, TripAdvisor would augment its services for travellers, granting enhanced control and flexibility, hence refining the planning experience for users in pursuit of effective and convenient travel solutions.

In addition, the absence of budget estimation and allocation features can be resolved by integrating a budgeting module that provides travellers with estimated spending ranges before travel and allows them to allocate funds across categories such as accommodation, food, and transportation. Adding real-time budget alerts would further improve financial control, helping users avoid overspending and increasing their confidence in managing expenses.

Finally, the lack of a similar-place substitution feature could be overcome by enabling the system to recommend alternative locations when the original choices are unavailable, overcrowded, or less appealing. By suggesting nearby and relevant options, TripAdvisor could ensure itineraries remain flexible, personalised, and practical, reducing the effort required from users to manually adjust their plans.

2.3 Comparision of existing travel management system above and the proposed system ExploreEasy

Features	Budget and Experience Based Travel Planner	Wanderlog	TripAdvisor	ExploreEasy(The proposed system)
AI-Generated Itineraries	Yes	Yes	Yes	Yes
Personalized Recommendations	Yes	Yes	Yes	Yes
Expense Tracking	Yes	Yes	No	Yes
Automatic Accommodation Suggestions	No	No	Yes	Yes
Weather Prediction Integration	No	No	No	Yes
Allow Budget Allocation	Yes	No	No	Yes
Alert User when Budget nearly exceed	No	No	No	Yes
Budget Range Estimation	No	No	No	Yes
Similar-place substitution feature	No	No	No	Yes

Table 2.3 Comparison of feature of other existing system to proposed system

Chapter 3
System Methodology/Approach

3.1 Methodologies and General Work Procedures

In this project, the methodology that will be used is Agile Methodology as software development on mobile devices is a perfect fit for agile. Agile methodologies use an iterative approach to quickly develop software, breaking down the entire software development lifecycle into smaller iterations. This reduces overall risk, enables quick adaptation to changes, eliminates the need for an upfront requirements freeze, and keeps the project on track and within budget. This methodology can be categorized into 6 phases, which are Concept, Inception, Development, Testing, Release, and Maintenance.

1. Concept

This phase involves defining the project's objectives and scopes such as the AI- driven trip planning and recommendation system, budget estimation and management feature and flexibility on choosing alternative places.

2. Inception

Finding every feature that can be included in this application is crucial during the inception stage. For example, the primary function, which is the AI-driven trip planning and recommendation system, should be prioritized. Furthermore, I have chosen Dart as the most suitable programming language and Flutter framework to complete my project.

3. Development

This phase will be the longest phase as all of the functions are needed to be completed during this phase, especially the main feature which is the AI-driven trip planning and recommendation system. For the other additional functions which are budget estimation and management feature, and flexibility on choosing alternative places will be completed iteratively after completion of the main feature.

4. Testing

The testing phase is conducted along with development phase. For example, the AI-driven trip planning and recommendation system is tested to ensure it matches user preferences.

5. Release

After completing development and testing all features, the application will be ready to release. Any comments or feedback from user will be collected for further improvement.

6. Maintenance

During the maintenance phase, users will receive a fully developed travel management application. Users' issues will be fixed during this stage to guarantee that every function is operating as intended.

3.2 Review of Technologies

3.2.1 Hardware Platform

The hardware involved in this project is computer and android mobile device.

Description	Specifications
Model	VivoBook_ASUSLaptop X513UA_M513UA
Processor	AMD Ryzen 7 5700U with Radeon Graphics, 1801 Mhz, 8 Core(s), 16 Logical Processor(s)
Operating System	Windows 10
Graphic	AMD Radeon Graphics
Memory	8.00 GB
Storage	475 GB

Table 3.2.1 Specifications of laptop

3.2.2 Operating System

This project is developed and tested on **Windows 10**, the primary operating system environment chosen due to its stability, compatibility, and developer support. Windows 10 provides a reliable platform for running essential development tools such as Flutter SDK, Android Studio, and Firebase integration utilities, all of which are crucial for building and testing the AI-driven travel management system.

By offering strong hardware driver support and compatibility with a wide range of frameworks, Windows 10 ensures smooth execution of emulators and debugging tools required for mobile app development. Its built-in features such as Windows Subsystem for Linux (WSL) and Hyper-V also allow flexibility for testing and virtualisation when needed. Furthermore, Windows 10's integration with cloud services (e.g., OneDrive) and its frequent updates enhance both security and performance, ensuring that the development environment remains up-to-date and protected.

In the context of this project, Windows 10 acts as the backbone operating system where all project activities—from coding and testing to documentation—are carried out. Its widespread adoption and support ecosystem make it a dependable choice for ensuring smooth project execution and consistent results.

3.2.3 Database

Firebase is a cloud-based platform by Google that provides a NoSQL database through Cloud Firestore, enabling real-time data storage and synchronisation. In this project, Firebase is used as the primary database to manage user profiles, trip details, expenses, budgets, and itineraries. Its real-time updates allow changes such as expense tracking or trip adjustments to be instantly reflected across devices, ensuring a smooth and collaborative experience for group travellers. With built-in support for authentication and scalability, Firebase offers a secure, efficient, and reliable database solution for the AI-driven travel management system.

3.2.4 Programming Language

This project primarily uses **Dart**, the programming language developed by **Google**, which serves as the foundation for the **Flutter framework**. Dart is an object-oriented, client-optimized language designed for building fast and scalable cross-platform applications. One of its main strengths is the ability to compile into native machine code, enabling high performance on both Android and iOS devices, as well as supporting web and desktop platforms. Dart also offers a clean syntax, asynchronous programming support through Futures and Streams, and hot-reload functionality when used in Flutter, significantly improving development efficiency. In the context of this project, Dart is used to implement the frontend and business logic of the AI-driven travel management system. All user interface components, trip planning workflows, budget management features, and integration with Firebase services are developed using Dart through Flutter. Its reactive framework allows for the creation of a modern and user-friendly interface while maintaining seamless communication with the backend database and APIs. By adopting Dart, this project benefits from a unified codebase, rapid prototyping, and performance consistency across platforms, making it an ideal choice for delivering a smooth and reliable user experience.

3.2.5 Summary of the Technologies Reviewed

In summary, the technologies selected for this project provide a robust and reliable foundation for developing the AI-driven travel management system. The hardware platform, consisting of an ASUS VivoBook laptop and an Android mobile device, offers sufficient processing power, memory, and storage capacity to support development, testing, and deployment. The **Windows 10** operating system ensures stability, compatibility, and developer support, enabling seamless execution of tools such as Flutter SDK, Android Studio, and Firebase integration utilities. For data management, **Firebase** serves as the cloud-based database solution, offering real-time synchronisation, scalability, and secure storage of user, trip, and budget-related information. Finally, **Dart** is adopted as the primary programming language, enabling the implementation of a cross-platform application through the Flutter framework, ensuring efficient development, performance consistency, and a modern user experience. Collectively, these technologies align to support the project’s objectives of delivering a smart, adaptive, and user-centric travel planning system.

3.3 System Design Diagrams

3.3.1 Block Diagram

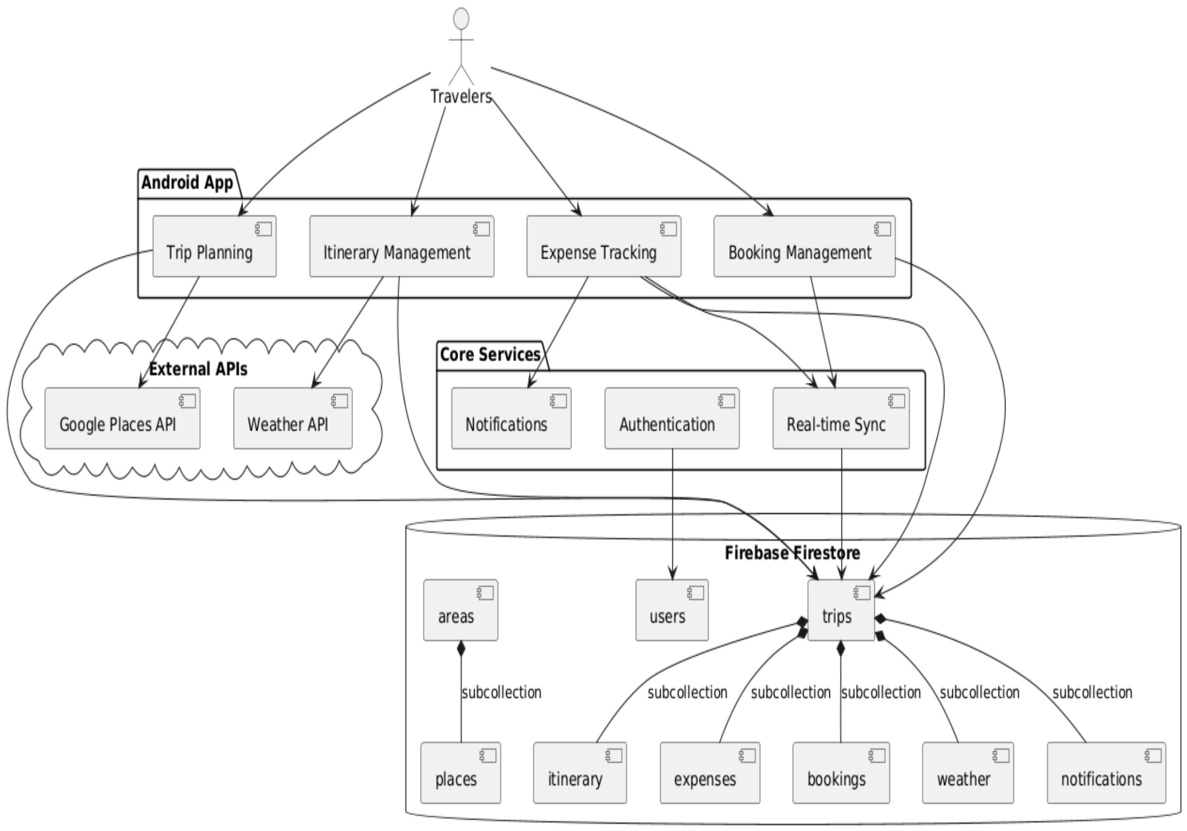


Figure 3.3.1 Block Diagram

The block diagram presents the system architecture of the Android-based travel management application, organized into five layers: users, application features, core services, database, and external APIs. At the top level, travelers are the end-users who directly interact with the Android app. The app is designed around four core features: Trip Planning, which allows users to specify destinations, dates, budgets, and preferences; Itinerary Management, which structures travel schedules and integrates with weather data to guide activity planning; Expense Tracking, which records transactions, manages group cost splitting, and connects with notifications to alert users when budgets are exceeded; and Booking Management, which handles accommodation reservations and uses real-time synchronization to keep all collaborators updated on booking status. Supporting these features are the Core Services, which operate in the background: Authentication secures user identity and controls access to shared trip data; Notifications deliver real-time alerts about expenses, bookings, and updates; and Real-time Sync ensures that booking inventory is updated immediately and expenses made and paid will be updated for group travelers too.

All data is persistently stored in Firebase Firestore, structured into top-level collections including users, trips, areas, and places. Within each trip document, specialized subcollections store related data such as itinerary, expenses, bookings, weather, and notifications. The areas collection also contains a places subcollection with detailed location attributes such as ratings and categories. To extend beyond core functionality, the system connects to external APIs. The Weather API provides forecast data to support itinerary planning, while the Google Places API enriches trip planning by supplying place details and recommendations. The arrows in the diagram illustrate data flows: expense tracking connects to notifications to trigger alerts, booking management connects to real-time sync for updates, and both itinerary management and trip planning connect to external APIs. Overall, the block diagram demonstrates how user-facing features, background services, database organization, and external integrations are coordinated to deliver a secure, collaborative, and intelligent travel management solution.

3.3.2 Use Case Diagram

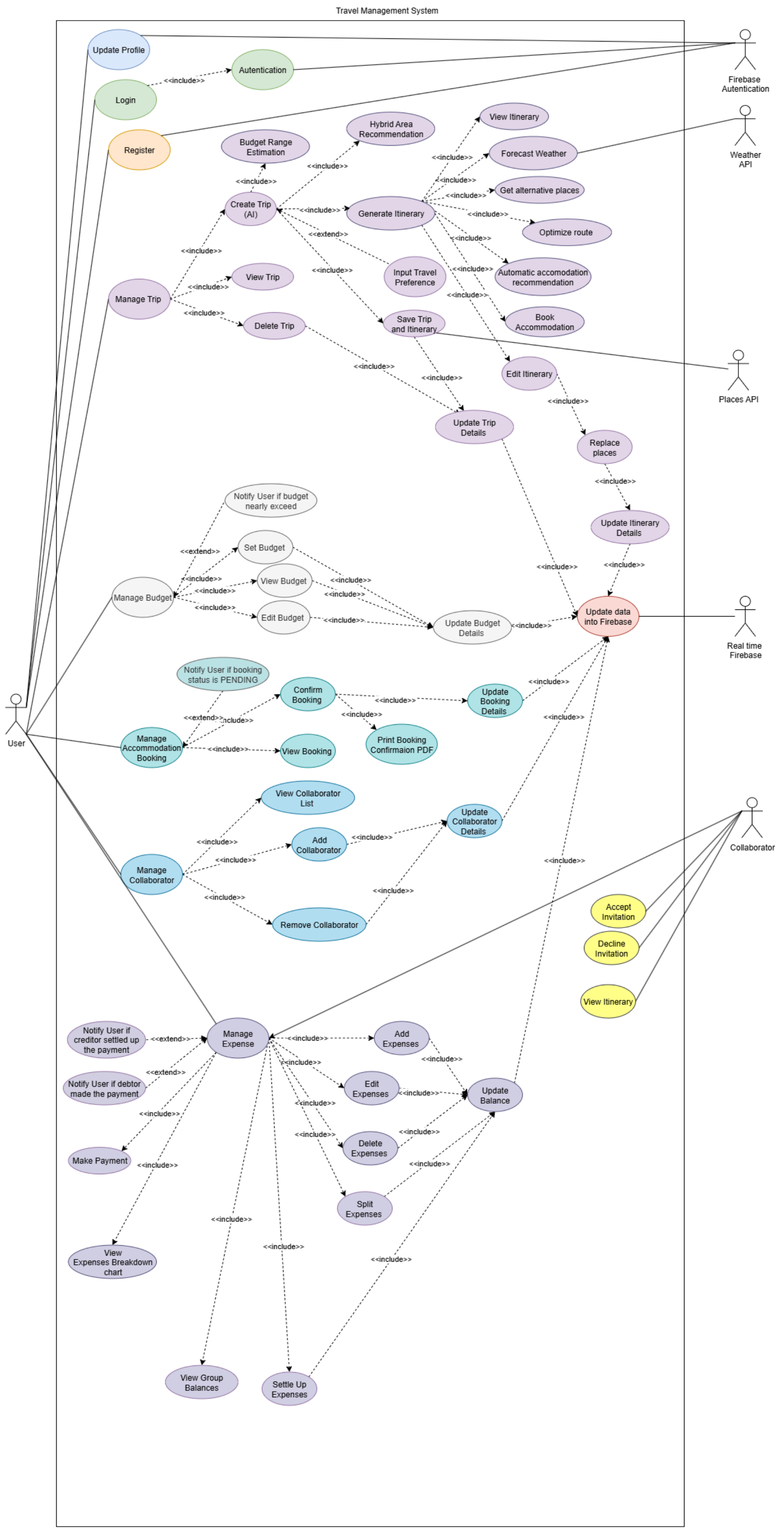


Figure 3.3.2 Use Case diagram

This Travel Management System use case diagram illustrates a comprehensive platform that helps users plan, book, and manage their travel experiences while handling expenses and budgets.

User Management and Authentication The system begins with basic user functionality where users can register for accounts, log in, and update their profiles. The authentication process connects to external services to verify user credentials, ensuring secure access to the platform.

At the core of the system is trip management functionality. Users can create new trips, view existing ones, and delete trips they no longer need. The system provides intelligent features like budget range estimation, hybrid area recommendations, and itinerary generation. Users can view routes that are being optimized automatically, get weather forecasts for their destinations, and receive automatic accommodation recommendations. The platform also suggests alternative places and allows users to save trips with their complete itineraries.

The system handles the booking process by allowing users to book accommodations directly through the platform. It manages booking statuses, sends confirmation notices when bookings are pending, and provides booking confirmation PDFs. Furthermore, users can view their booking details.

The platform supports collaborative trip planning where users can work together on trips. The system manages collaborators by allowing owner to add, view, and remove collaborators from their trips. Collaborators can accept or decline invitations and view itineraries for trips they're involved in.

A robust expense management system tracks trip-related costs. Users can add, edit, delete, and split expenses among travel companions. The system provides expense breakdowns, updates account balances, and allows users to view group balances and settle expenses. Users receive notifications when payments are made or when creditors need to be notified about expense settlements.

The system connects to several external APIs to enhance functionality: a Weather API provides forecast information, a Google Places API helps with location services and recommendations, and a Real-time Feedback system keeps users informed about their trip status and updates.

This comprehensive system essentially serves as a one-stop solution for travelers, combining trip planning, booking management, expense tracking, and collaborative features all in one platform.

3.3.3 Activity Diagram

3.3.3.1 – Generate Trip

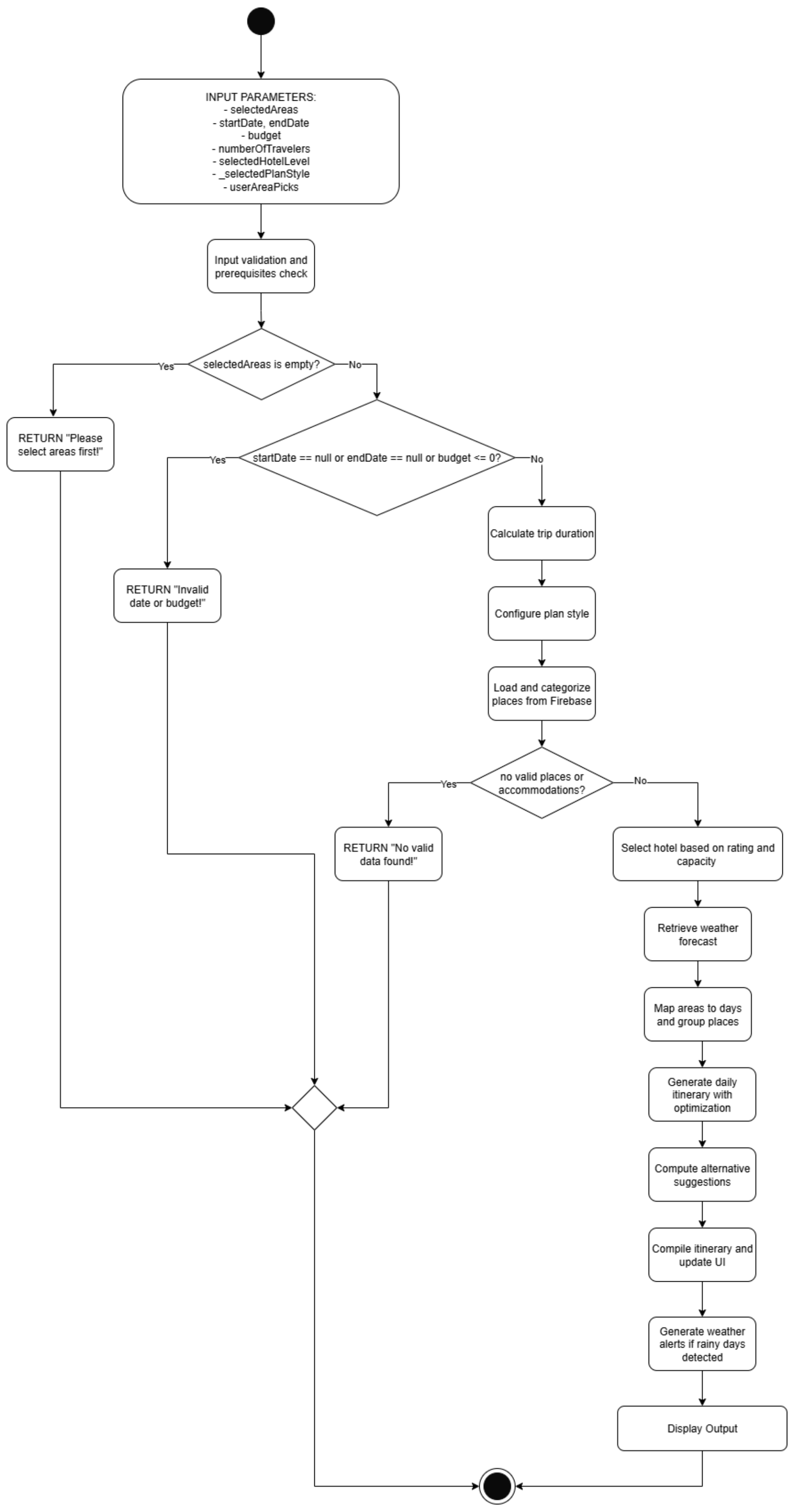

INPUT PARAMETERS:
- selectedAreas
- startDate, endDate
- budget
- numberOfTravelers
- selectedHotelLevel
- _selectedPlanStyle
- userAreaPicks
Input validation and prerequisites check
selectedAreas is empty?
Yes
No
RETURN "Please select areas first!"
startDate == null or endDate == null or budget <= 0?
Yes
No
RETURN "Invalid date or budget!"
Calculate trip duration
Configure plan style
Load and categorize places from Firebase
no valid places or accommodations?
Yes
No
RETURN "No valid data found!"
Select hotel based on rating and capacity
Retrieve weather forecast
Map areas to days and group places
Generate daily itinerary with optimization
Compute alternative suggestions
Compile itinerary and update UI
Generate weather alerts if rainy days detected
Display Output

Figure 3.3.3.1 Activity Diagram – Generate Trip

This activity diagram shows the "Generate Trip" process for a travel management system. The process begins when a user provides input parameters including selected areas, start and end dates, budget, number of travelers, hotel level preferences, plan style, and area picks. The system first validates these inputs and checks prerequisites.

The flow then checks if any areas were selected. If no areas are chosen, the system prompts the user to "Please select areas first" and ends the process. If areas are selected, the system validates the dates and budget - if any are invalid (null dates or zero budget), it returns an error message. When all inputs are valid, the system calculates the trip duration and configures the appropriate plan style.

Next, the system loads and categorizes places from Firebase (likely a database of destinations and accommodations). If no valid places or accommodations are found, it returns a "No valid data found" message. However, if data is available, the system proceeds with the trip generation by selecting hotels based on rating and capacity, retrieving weather forecasts, and mapping areas to specific days while grouping places logically.

The system then generates a daily itinerary with optimization features, computes alternative suggestions, and compiles everything into a comprehensive itinerary that updates the user interface. Finally, it generates weather alerts for any rainy days detected during the trip period. The output includes detailed daily names and points with route data, selected hotel information with alternatives, alternative places like restaurants and attractions, trip prices stored in the database, weather data and alerts, and UI widgets displaying the complete map and itinerary.

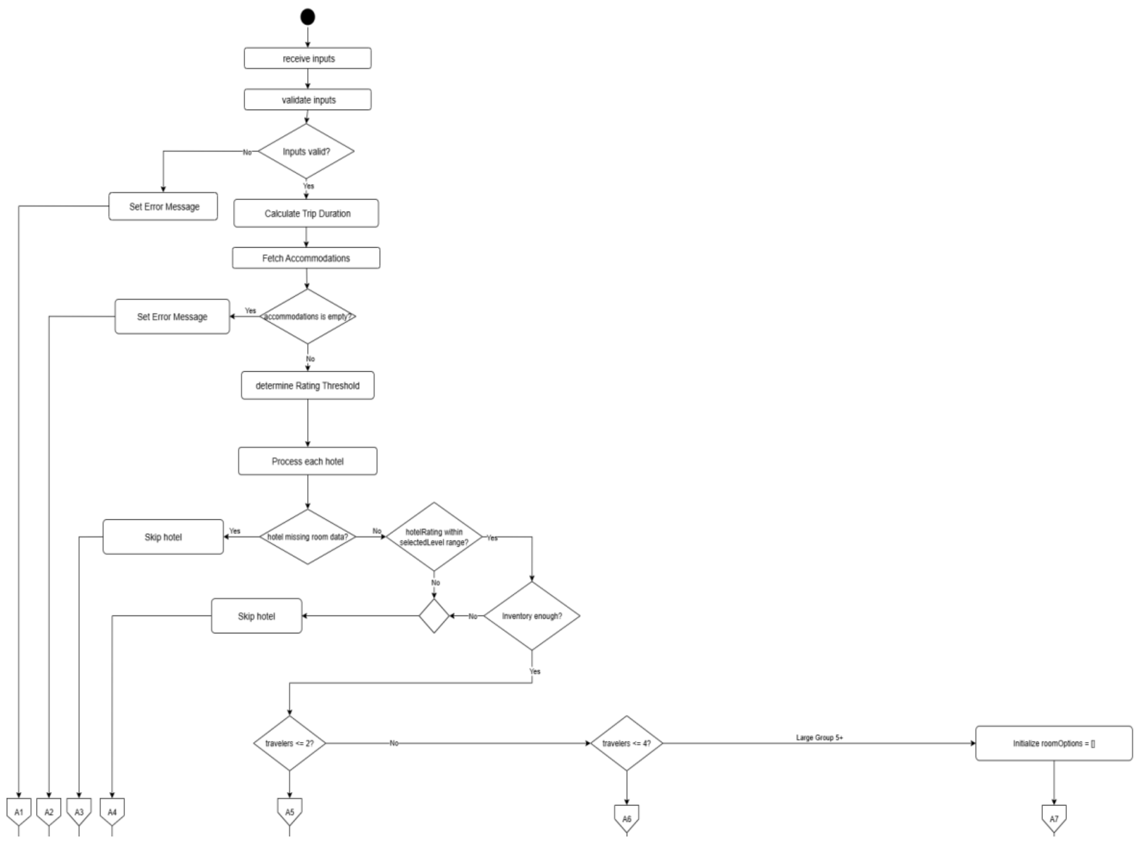
Use Case Description

Field	**Description**
Use Case Name	Generate Trip
Actor	User (Traveler)
Description	System generates a personalized trip itinerary based on user preferences including destinations, dates, budget, and travel style
Preconditions	- User must be logged into the system - User must have selected at least one travel area - System must have access to places database (Firebase) - Weather API must be accessible
Postconditions	- Complete trip itinerary is generated and displayed - Hotel recommendations are provided - Weather alerts are generated if needed

	- Trip data is stored in database - User interface is updated with trip details
Main Flow	1. User provides trip parameters (areas, dates, budget, travelers, hotel level, plan style) 2. System validates input parameters 3. System calculates trip duration 4. System configures plan style 5. System loads places from database 6. System selects appropriate hotels 7. System retrieves weather forecast 8. System maps areas to daily schedule 9. System generates optimized itinerary 10. System computes alternative suggestions 11. System compiles final itinerary 12. System generates weather alerts 13. System displays complete trip plan
Alternative Flows	**Alt 1**: No areas selected - System prompts user to select areas first **Alt 2**: Invalid dates/budget - System returns error message **Alt 3**: No valid places found - System returns "No valid data found" message
Exceptions	- Database connection failure - Weather API unavailable - Invalid input parameters - No accommodations available for selected criteria

Table 3.3.3.1 Use Case Description – Generate Trip

3.3.3.2 – Automated Accommodation Suggestion



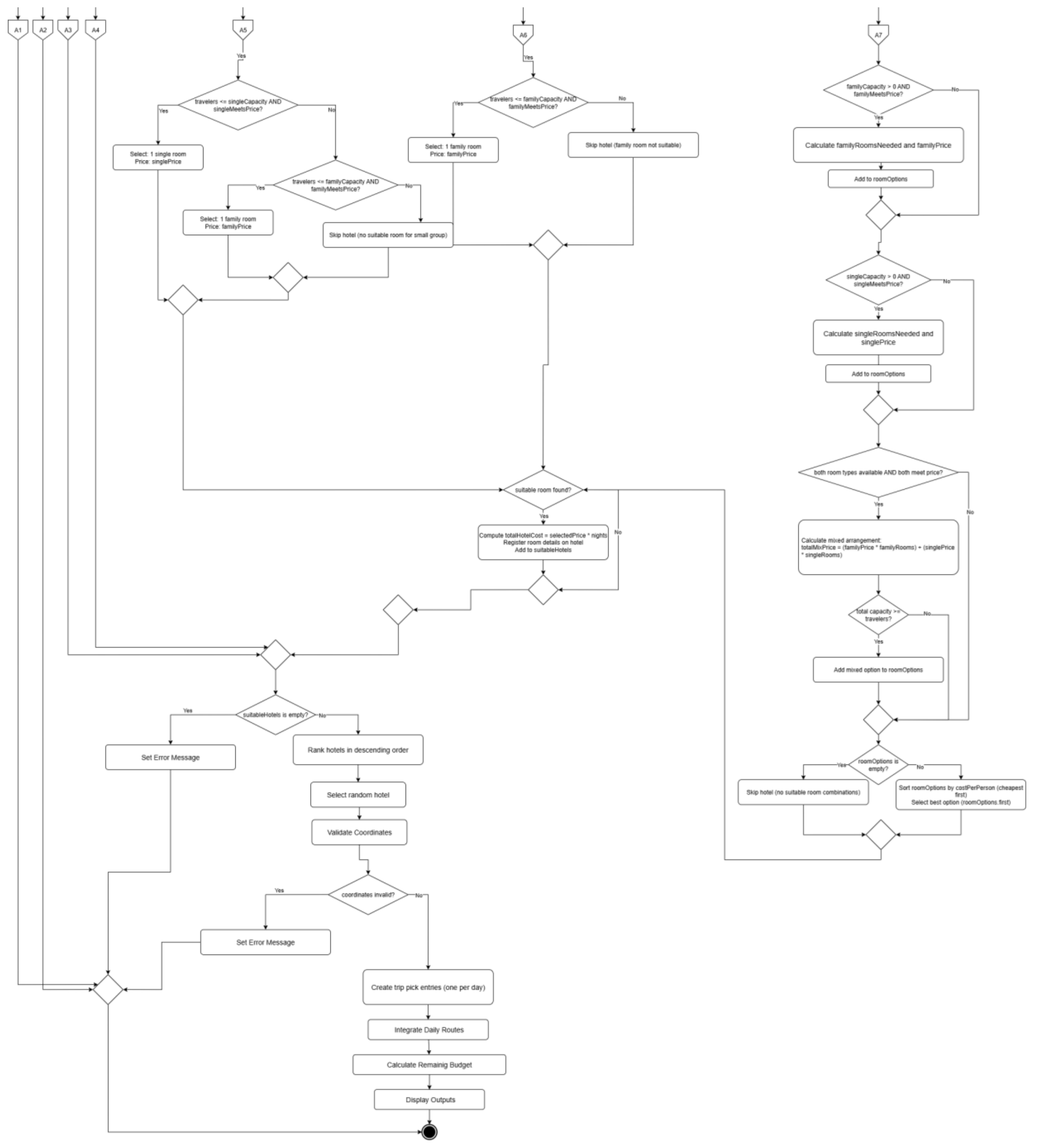

A1
A2
A3
A4
A5
Yes
travelers <= singleCapacity AND singleMeetsPrice?
Yes
No
Select: 1 single room Price: singlePrice
travelers <= familyCapacity AND familyMeetsPrice?
Yes
No
Select: 1 family room Price: familyPrice
Skip hotel (no suitable room for small group)
A6
Yes
travelers <= familyCapacity AND familyMeetsPrice?
Yes
No
Select: 1 family room Price: familyPrice
Skip hotel (family room not suitable)
A7
familyCapacity > 0 AND familyMeetsPrice?
No
Yes
Calculate familyRoomsNeeded and familyPrice
Add to roomOptions
singleCapacity > 0 AND singleMeetsPrice?
No
Yes
Calculate singleRoomsNeeded and singlePrice
Add to roomOptions
both room types available AND both meet price?
Yes
No
Calculate mixed arrangement: totalMixPrice = (familyPrice * familyRooms) + (singlePrice * singleRooms)
total capacity >= travelers?
No
Yes
Add mixed option to roomOptions
roomOptions is empty?
Yes
No
Skip hotel (no suitable room combinations)
Sort roomOptions by costPerPerson (cheapest first) Select best option (roomOptions.first)
suitable room found?
Yes
No
Compute totalHotelCost = selectedPrice * nights Register room details on hotel Add to suitableHotels
suitableHotels is empty?
Yes
No
Set Error Message
Rank hotels in descending order
Select random hotel
Validate Coordinates
coordinates invalid?
Yes
No
Set Error Message
Create trip pick entries (one per day)
Integrate Daily Routes
Calculate Remainig Budget
Display Outputs

Figure 3.3.3.2 Activity Diagram – Automated Accommodation Suggestion

The accommodation selection process begins when the system receives and valiates input parameters including selected travel areas, travel dates, number of travelers, hotel level preference, and budget. After validation, the system calculates the trip duration by determining the number of nights between the start and end dates, ensuring a minimum of one night even for same-day bookings.

The system then retrieves hotel data from the database for all selected areas. If no hotels are found in the specified locations, the process terminates with an appropriate error message. Otherwise, the system proceeds to evaluate each hotel individually.

For each hotel, the system first checks whether the hotel's rating meets the requirements for the user's selected hotel level. Hotels that don't meet the minimum rating standards are immediately skipped. For qualifying hotels, the system will check inventory availability for the user’s chosen dates. If it is available, the system then enters the smart room selection phase, which is the core innovation of this process.

The room selection logic varies significantly based on group size. For small groups of 1-2 travelers, the system prioritizes single rooms as they are typically more economical. If a suitable single room that meets both capacity and price requirements is available, it's selected. If not, the system falls back to family rooms as an alternative. This approach ensures cost optimization for smaller groups while maintaining flexibility.

Medium groups of 3-4 travelers follow a different strategy. The system directly looks for family rooms that can accommodate the entire group. This eliminates the complexity of multiple room bookings while ensuring everyone stays together. The family room must meet both capacity requirements and the minimum price standards for the selected hotel level.

Large groups of 5 or more travelers require the most sophisticated processing. The system calculates multiple room combination scenarios including all family rooms, all single rooms, and mixed arrangements combining both room types. For each viable combination, it calculates the total cost and cost per person. The system then selects the most economical option that can accommodate all travelers while meeting quality standards.

Throughout the room selection process, the system validates that room prices meet the minimum standards for the selected hotel level. This ensures that budget-conscious selections don't compromise on quality expectations. Hotels or room arrangements that don't meet these standards are excluded from consideration.

Once all suitable hotels have been identified and their optimal room arrangements determined, the system ranks them by hotel rating. To provide variety and prevent predictable selections, the system randomly chooses from the top-rated options rather than always selecting the highest-rated hotel.

The final phase involves cost calculation and budget allocation. The system computes the total accommodation cost based on the selected hotel and room arrangement, then determines how much budget remains for activities and other trip expenses. This information is crucial for the overall trip planning process.

The process concludes by updating the trip itinerary with the selected hotel as the base location and providing comprehensive output including hotel details, room arrangement, cost breakdown, and budget allocation. This ensures that all subsequent trip planning activities can reference the confirmed accommodation as the starting point for daily itineraries.

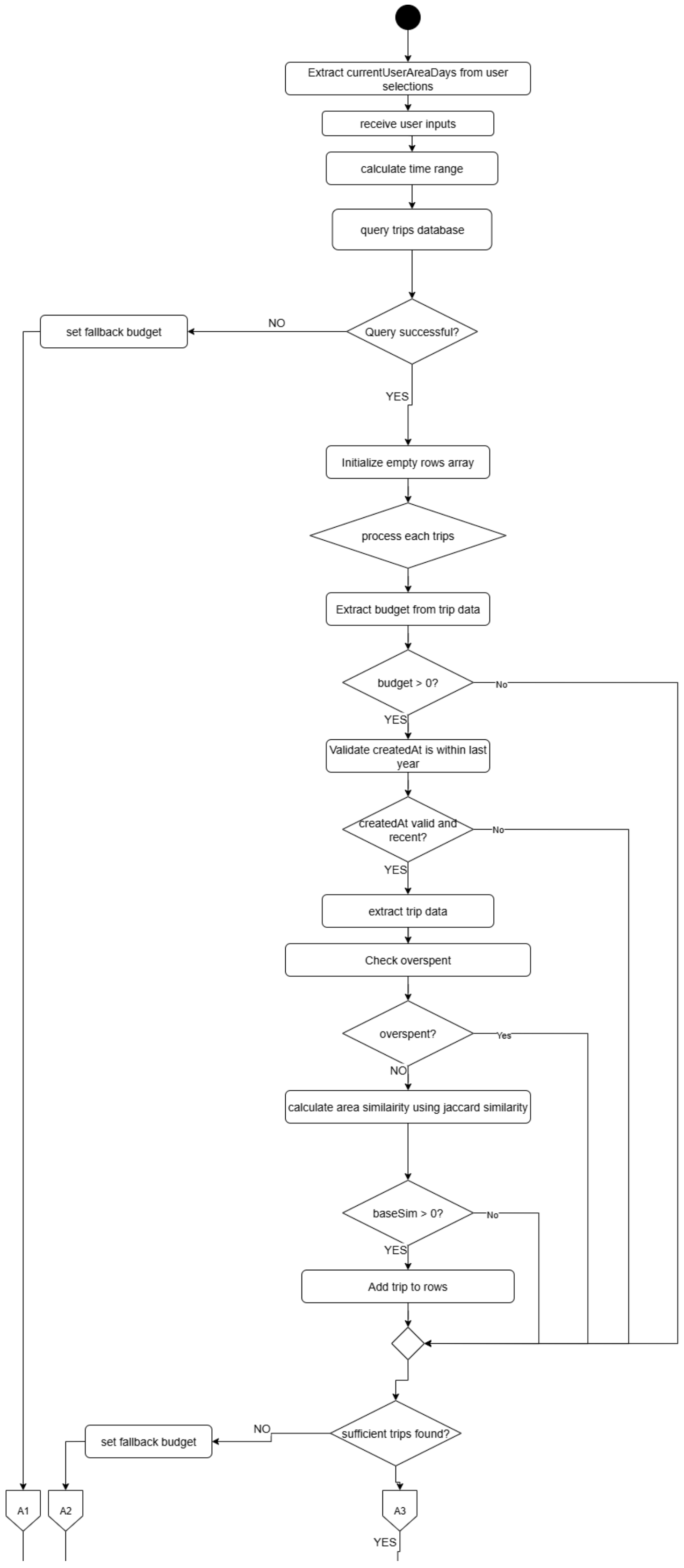
Use Case Description

Field	**Description**
Use Case Name	Automated Accommodation Suggestion
Actor	User (Traveler)
Description	System automatically suggests suitable accommodations based on user preferences, location, budget, and availability, with options for booking and alternative recommendations
Preconditions	- User must be logged into the system - User travel preferences must be set - Accommodation database must be accessible - Location and date parameters must be provided - Budget constraints must be specified
Postconditions	- List of suitable accommodations is generated - Accommodation details are displayed to user - Booking options are made available - Alternative suggestions are provided if needed - User selection is recorded in the system
Main Flow	1. System collects user accommodation preferences (location, dates, budget, type) 2. System validates input parameters 3. System searches accommodation database based on criteria 4. System filters results by availability and budget 5. System ranks accommodations by relevance and rating 6. System presents top accommodation suggestions 7. User reviews accommodation options 8. System provides detailed accommodation information 9. System offers booking functionality 10. System records user interaction and preferences
Alternative Flows	**Alt 1**: No accommodations found - System suggests relaxing criteria or alternative locations

	Alt 2: Budget constraints too restrictive - System suggests budget-friendly alternatives **Alt 3**: Dates unavailable - System suggests alternative dates or similar properties **Alt 4**: Location not supported - System suggests nearby supported areas
Exceptions	- Accommodation database unavailable - Invalid location or date parameters - Network connectivity issues External booking system failures - No accommodations match criteria

Table 3.3.3.2 Use Case Description – Automated Accommodation Suggestion

3.3.3.3 – Budget Range Estimation



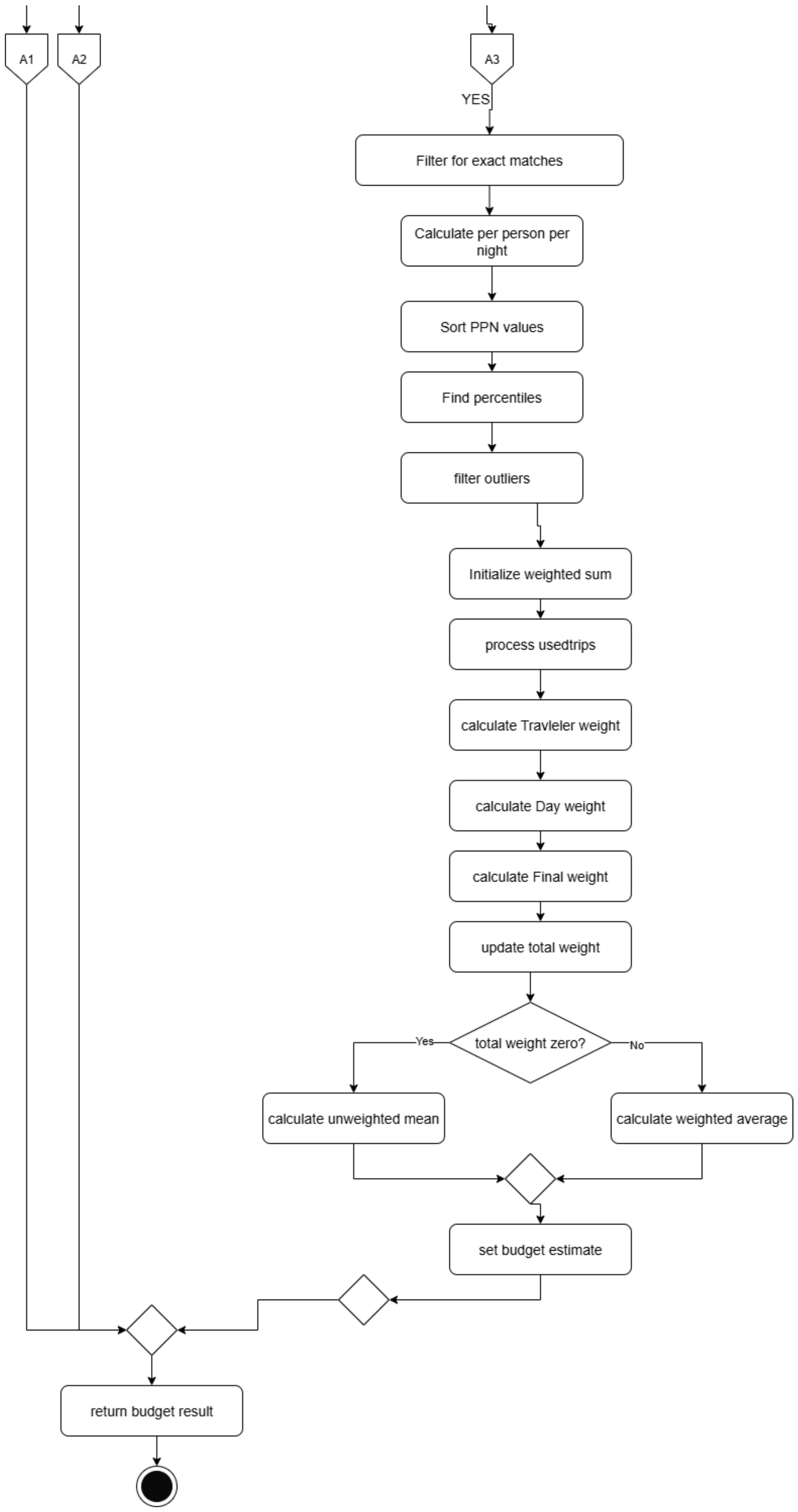


Figure 3.3.3.3 Activity Diagram – Budget Range Estimation

The activity diagram illustrates the process of estimating a suitable budget range for a user based on previous trip data and user inputs. The process begins by extracting the user's selected areas and days, then receiving user inputs and calculating the trip's time range. A query is executed on the trips database, and if the query fails, the system falls back to a default budget. If successful, the system initializes an array and processes each trip.

For each trip, the budget is extracted and validated to ensure it is greater than zero. The trip creation date is then checked to confirm it falls within the past year. If valid, the trip data is extracted, overspending is verified, and area similarity is calculated using Jaccard similarity. Trips with a base similarity greater than zero are added into the collection. If insufficient trips are found, the system again sets a fallback budget. Otherwise, it proceeds with detailed filtering.

The system filters for exact matches, calculates the per-person-per-night (PPN) values, sorts them, and determines percentiles. Outliers are removed, and a weighted sum process begins. Each used trip is processed, calculating traveler weight, day weight, and final weight, which are then combined into the total weight. Depending on whether the total weight is zero, the system calculates either an unweighted mean or a weighted average. Finally, the budget estimate is set and returned to the user as the result.

This workflow ensures that the budget estimation is data-driven, adaptive, and reflective of recent, relevant trip information, with robust fallback mechanisms in case of insufficient or invalid data.

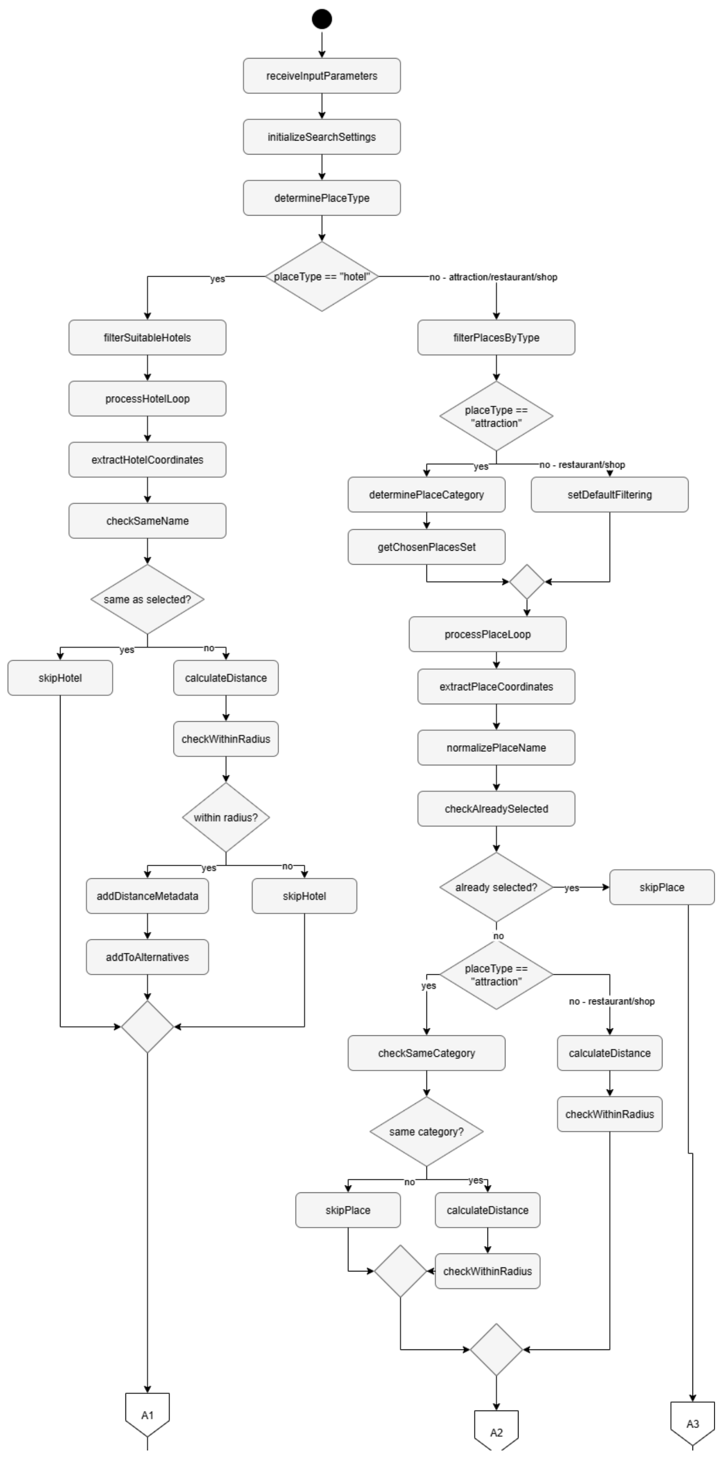
Use Case Description

Field	**Description**
Use Case Name	Estimate Budget Range
Actor	User (Traveler)
Description	The system estimates a realistic budget range for the user's trip based on previous trips stored in the database, user selections, and similarity analysis.
Trigger	User selects trip details (area, days, inputs) and requests a budget estimate.
Preconditions	- User inputs (area, days, travelers) must be available. - Trips database must be accessible. - Past trips with valid budget data exist (else fallback applies).
Postconditions	- Budget range estimate is generated.

	- If insufficient data, fallback budget is returned. - Budget estimate is displayed to the user.
Main Flow	1. Extract user area, days, and inputs. 2. Calculate trip time range. 3. Query database for past trips. 4. Process each trip (validate budget, date, overspending). 5. Calculate area similarity using Jaccard. 6. Collect valid trips and filter exact matches. 7. Calculate per-person-per-night(PPN) values. 8. Remove outliers and apply weighted calculations. 9. Estimate and return final budget.
Alternative Flow	- 3a. Query fails → set fallback budget. - 5a. No sufficient trips → set fallback budget. - 8a. Total weight = 0 → calculate unweighted mean instead of weighted average.
Exceptions	- Trips with budget ≤ 0 are ignored. - Trips older than 1 year are excluded. - Overspent trips are discarded.

Table 3.3.3.3 Use Case Description – Budget Range Estimation

3.3.3.4 – Get Alternative Places



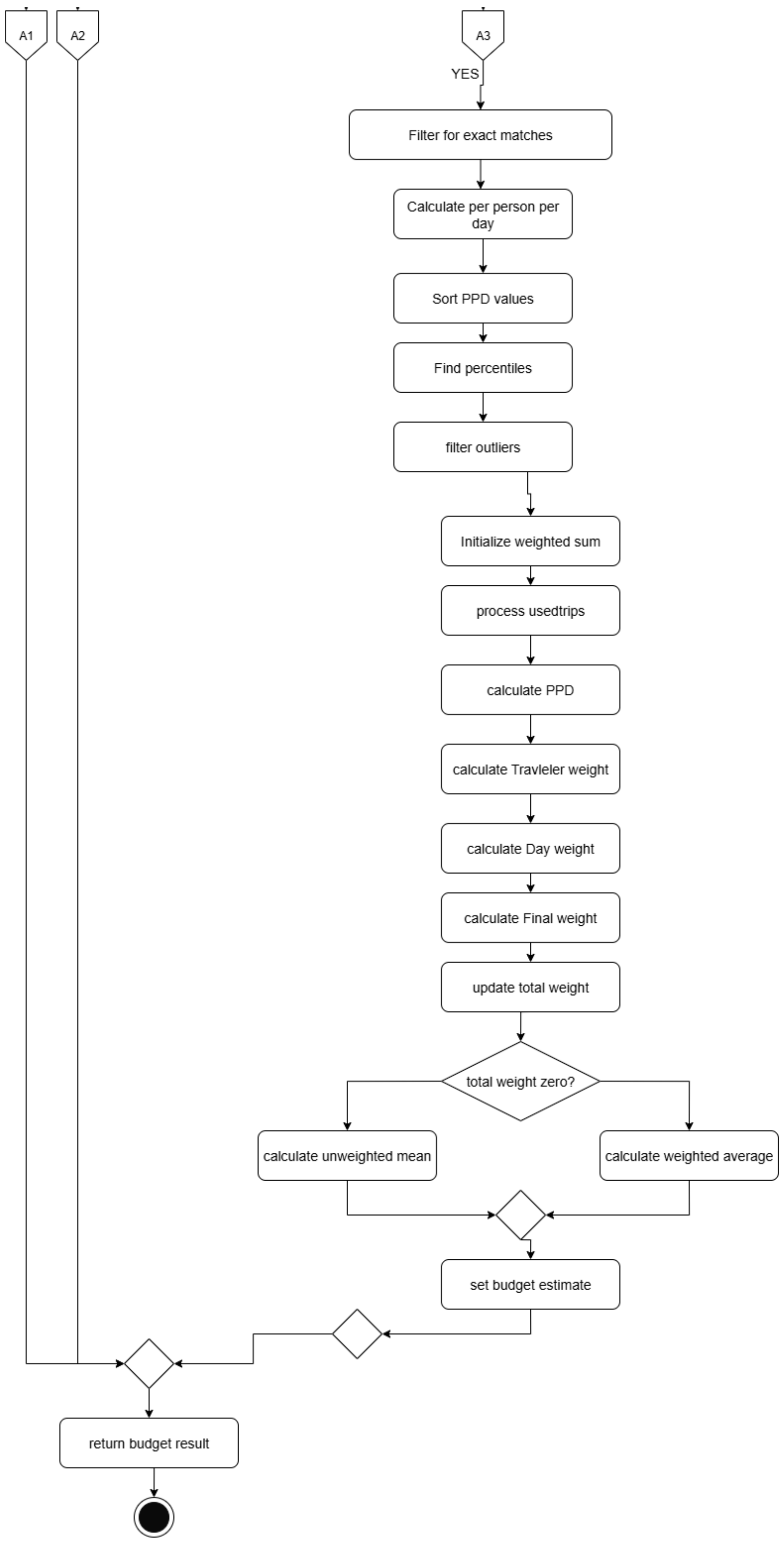


Figure 3.3.3.4 Activity Diagram – Get Alternative Places

The get alternative places system provides users with nearby location options when they want to replace a selected place in their travel itinerary. The process begins when the system receives input parameters including the current location coordinates, place type, selected place name, and search radius. The system then initializes search settings by determining appropriate radius limits based on the place type, with all place types using 2.0 km as their default search boundary.

The core processing splits into two distinct paths based on place type. Hotels follow a specialized path because they utilize a pre-filtered collection called suitableHotels, which contains accommodations that have already passed rating, capacity, and budget requirements during the initial trip generation process. The system filters through this collection, processes each hotel individually, and extracts coordinates before checking if the candidate hotel has the same name as the currently selected one. If the names differ, the system calculates the distance using the Haversine formula and verifies whether the hotel falls within the 2 km search radius. Qualifying hotels receive distance metadata and get added to the alternatives list. The second path handles attractions, restaurants, and shops through a unified process that begins with filtering places by type from the complete places database. This filtering applies type-specific algorithms to identify relevant candidates from the comprehensive collection of all available locations. For attractions, the system performs additional category determination to identify whether the selected place is a museum, temple, park, gallery, or other specific type, ensuring that alternatives match the same subcategory. Restaurants and shops bypass this category matching step through default filtering since any restaurant can generally substitute for another restaurant, and any souvenir shop can replace another gift shop. After type-specific processing, the system retrieves the chosen places set, which contains all locations already selected in the current itinerary to prevent duplicate suggestions. The main processing loop then begins, where each candidate place undergoes coordinate extraction, name normalization, and duplicate checking against the existing itinerary. For attractions, the system verifies that candidates belong to the same category as the selected place, while restaurants and shops proceed directly to distance calculations. The system calculates precise distances using the Haversine formula and checks whether each candidate falls within the appropriate radius threshold for its type.

Both processing paths converge at the sorting and limiting phase, where the system organizes all found alternatives by distance in ascending order, placing the closest options first. The system then limits results to a maximum of six alternatives to prevent user overwhelm while maintaining quality choices. Finally, the system returns the complete alternatives array through the single endpoint.

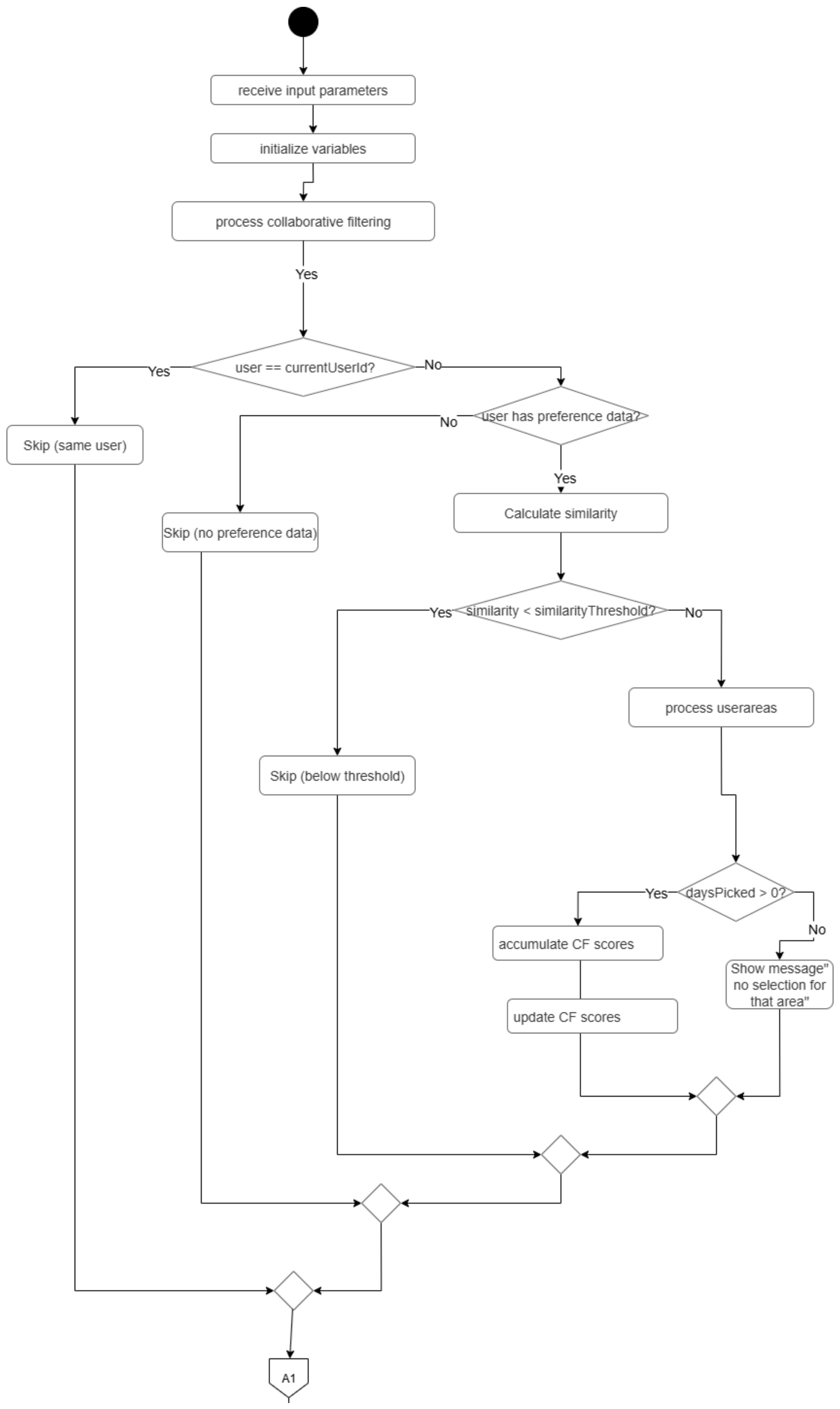
Use Case Description

Element	**Description**
Use Case Name	Get Alternative Places
Actor	User
Goal	Find nearby alternative places of the same type to replace a selected location in travel itinerary
Preconditions	• User has an existing itinerary with selected places • User wants to replace a specific place • Places database contains relevant alternatives • Current location coordinates are available
Input Parameters	• currentLocation: LatLng coordinates • placeType: String ("hotel", "attraction", "restaurant", "shop") • selectedPlaceName: String (current place name) • radius: Double (search radius in km)
Main Flow	1. System receives input parameters 2. System initializes search settings with 2.0km radius for all place types 3. System determines place type and routing path 4. For Hotels: Filter from pre-filtered suitableHotels collection, check distance within 2km 5. For Others: Filter places by type from complete database, perform category determination (attractions only) 6. System retrieves chosen places set to prevent duplicates 7. System processes each candidate with coordinate extraction and name normalization 8. System performs duplicate checking and category matching (attractions) 9. System calculates distances using Haversine formula and verifies radius constraints 10. System sorts results by distance and limits to 6 alternatives 11. System returns complete alternatives array through single endpoint
Alternative Flows	A1: No alternatives found - System returns empty array

	A2: All candidates too far - System skips places beyond 2km radius, returns available alternatives A3: All candidates already selected - System skips duplicates from existing itinerary, returns remaining alternatives A4: Wrong attraction category - System skips attractions not matching selected subcategory

Table 3.3.3.4 Use Case Description – Get Alternative Places

3.3.3.5 – Hybrid Area Recommendations



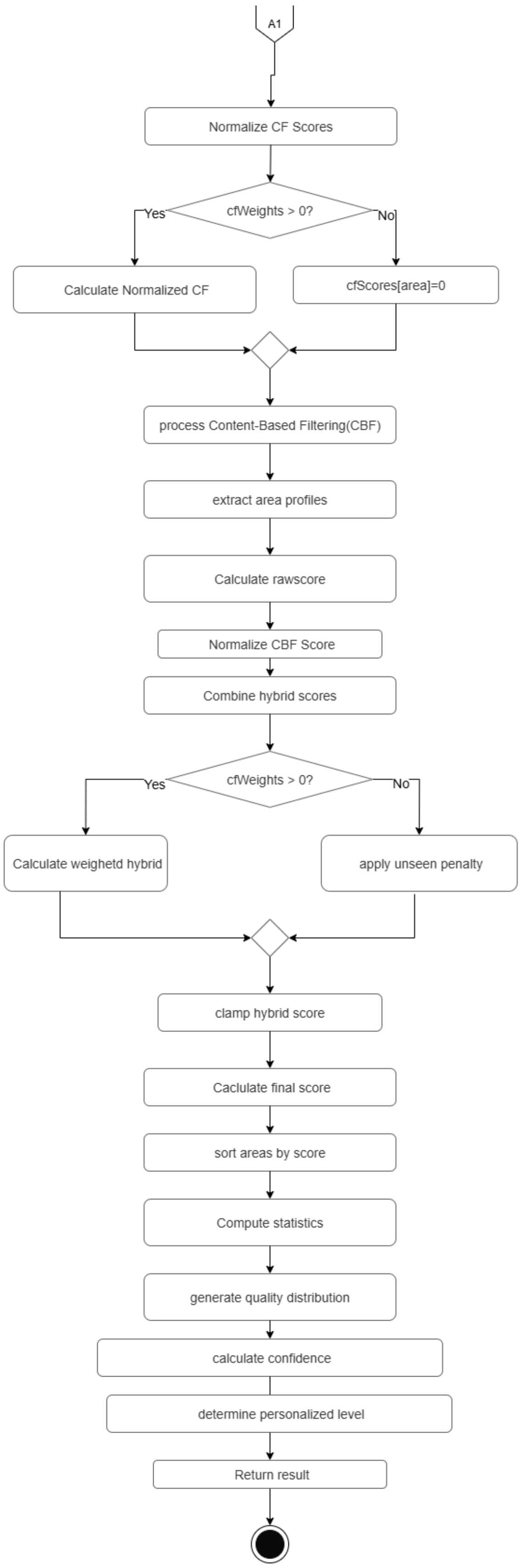


Figure 3.3.3.5 Activity Diagram – Hybrid Area Recommendations

This activity diagram shows an intelligent area recommendation system that helps users discover suitable travel destinations in Penang based on their preferences and behavior patterns. The process begins by collecting comprehensive input parameters including newUserPrefs which represents the current user's stated preferences and interests, userAreaPicks containing the user's selected or preferred areas from previous interactions, userPrefs which holds historical user preference data and ratings, penangAreas representing all available destinations in the Penang region, areaProfiles containing scoring profiles for each area based on characteristics like beach, city, nature, and history where each area has quantified scores for different attributes, and currentUserId which serves as the identifier for the current user requesting recommendations.

The system initializes key components including user preference vectors, similarity calculation algorithms, and hybrid recommendation weights using this input data. First, the system checks if the user has sufficient historical data stored in userPrefs and userAreaPicks. If the user has a rich history of interactions and ratings, the system can generate personalized recommendations based on past behavior patterns and preferences. However, if no substantial historical data exists for the user, the system falls back to content-based filtering using the user's stated preferences from newUserPrefs.

For users with sufficient historical data, the system implements collaborative filtering by calculating similarity scores with other users who have demonstrated similar travel patterns and preferences. The collaborative filtering calculation begins by creating user-area rating matrices from userPrefs and userAreaPicks data. The system then identifies users who have rated similar areas and calculates cosine similarity between the current user and other users using the formula: Similarity equals the dot product of User A ratings and User B ratings divided by the product of their magnitudes. After identifying the top K similar users as neighbors, the system generates recommendations based on areas that were highly rated by these similar users. The prediction for how much a user might like a specific area is calculated by taking the weighted sum of similarity scores multiplied by neighbor ratings, divided by the sum of all similarity scores.

Simultaneously, the system performs content-based filtering analysis by matching user preferences with area characteristics using the detailed areaProfiles data. The content-based filtering process starts by extracting the user profile vector from newUserPrefs, converting stated preferences into a numerical preference vector containing scores for beach preference, city preference, nature preference, and history preference. Each area in penangAreas has corresponding quantified scores in areaProfiles, creating an area profile vector with beach score, city score, nature score, and history score. The system then calculates content similarity between the user profile and each area profile using cosine similarity, which measures how closely the user's preferences align with what each area offers. Additional weighted scoring is applied based on the user's stated importance of each characteristic, ensuring that more important preferences have greater influence on the recommendations.

The system combines both collaborative filtering and content-based filtering recommendations using a sophisticated hybrid approach. The final score for each area is calculated using a weighted combination where the collaborative filtering score is multiplied by weight alpha and the content-based filtering score is multiplied by weight beta, with alpha plus beta equaling one. The weight distribution is dynamically adjusted based on the user's data availability - users with rich historical data receive higher alpha weights favoring collaborative filtering, while new users receive higher beta weights emphasizing content-based filtering. This ensures that both experienced users with substantial interaction history and new users with limited data receive relevant and personalized recommendations.After generating the hybrid scores, the system normalizes all scores to ensure fair comparison and ranks areas by their relevance to the user's preferences and patterns. The system concludes by formatting the output as a comprehensive ranked list of recommended areas, complete with confidence scores indicating the system's certainty in each recommendation.

Use Case Description

Field	**Description**
Use Case Name	Hybrid area recommendation
Actor	User (Traveler)
Description	System generates personalized travel area recommendations by combining collaborative filtering, content-based filtering, and user preference analysis to suggest destinations that match user interests and travel patterns
Preconditions	- User must be registered in the system - User preferences must be collected (interests, budget, travel style) - Area database with characteristics must be available - User similarity algorithms must be initialized - Recommendation engine must be operational
Postconditions	- Personalized list of recommended areas is generated - Recommendations include confidence scores and explanations - Areas are ranked by relevance to user preferences

	- Additional metadata (costs, best times to visit) is provided - User interaction data is recorded for future improvements
Main Flow	1. System collects user input parameters (preferences, budget, dates, group size) 2. System initializes recommendation algorithms and user vectors 3. System checks if user has sufficient historical travel data 4. **If historical data exists**: System performs collaborative filtering by finding similar user 5. **If no historical data**: System uses content-based filtering with stated preferences 6. System calculates area-user compatibility scores 7. System combines collaborative and content-based recommendations using hybrid weights 8. System applies contextual filters (budget, season, distance) 9. System normalizes and ranks recommendations by relevance 10. System formats output with explanations and metadata 11. System presents ranked list of recommended areas to user
Alternative Flows	**Alt 1**: Insufficient user data - System uses popular destinations and content-based filtering **Alt 2**: No areas match criteria - System relaxes constraints and suggests broader options **Alt 3**: User has very specific preferences - System prioritizes content-based over collaborative filtering **Alt 4**: Limited budget constraints - System filters recommendations by cost-effectiveness
Exceptions	- User preference data incomplete or invalid - Area database unavailable - Recommendation algorithms fail to initialize - Network connectivity issues - Insufficient computational resources for complex calculations - User similarity calculation errors

Table 3.3.3.5 Use Case Description – Hybrid Area Recommendations

3.3.3.6 – Weather Forecast

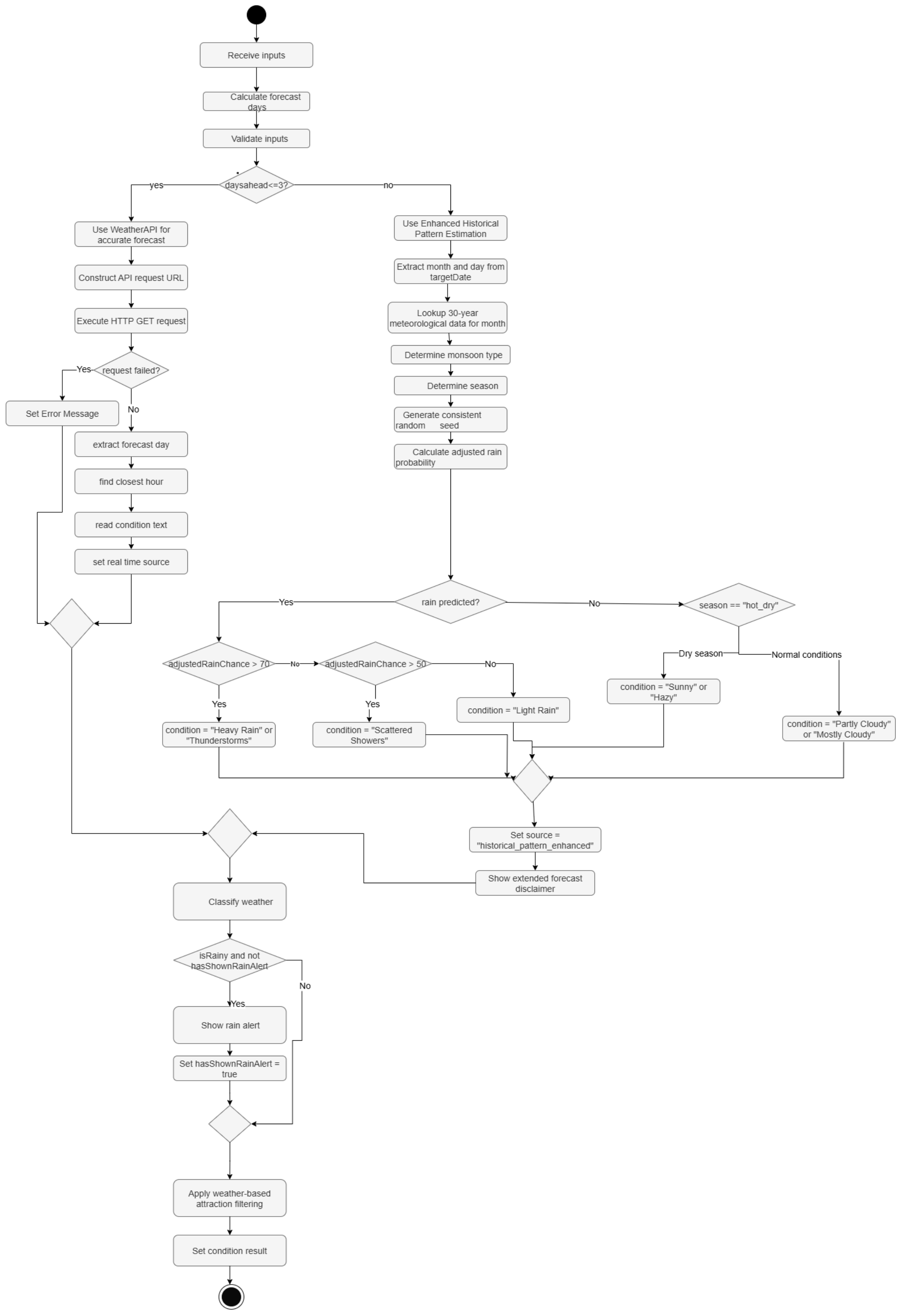

Receive inputs
Calculate forecast days
Validate inputs
daysahead<=3?
yes
no
Use WeatherAPI for accurate forecast
Construct API request URL
Execute HTTP GET request
request failed?
Yes
No
Set Error Message
extract forecast day
find closest hour
read condition text
set real time source
Use Enhanced Historical Pattern Estimation
Extract month and day from targetDate
Lookup 30-year meteorological data for month
Determine monsoon type
Determine season
Generate consistent random seed
Calculate adjusted rain probability
rain predicted?
Yes
No
season == "hot_dry"
Dry season
Normal conditions
adjustedRainChance > 70
No
adjustedRainChance > 50
No
Yes
Yes
condition = "Heavy Rain" or "Thunderstorms"
condition = "Scattered Showers"
condition = "Light Rain"
condition = "Sunny" or "Hazy"
condition = "Partly Cloudy" or "Mostly Cloudy"
Set source = "historical_pattern_enhanced"
Show extended forecast disclaimer
Classify weather
isRainy and not hasShownRainAlert
Yes
No
Show rain alert
Set hasShownRainAlert = true
Apply weather-based attraction filtering
Set condition result

Figure 3.3.3.6 Activity Diagram – Weather Forecast

The activity diagram illustrates the decision-making process for generating weather forecasts used in trip planning. The workflow begins with receiving user inputs, calculating the number of forecast days ahead, and validating the inputs. If the requested forecast date falls within three days, the system relies on the WeatherAPI for accurate real-time forecasts. It constructs an API request, executes the HTTP GET request, and, if successful, extracts the forecast for the relevant day and closest hour, reads the condition text, and sets the source to real-time weather data. If the API request fails, an error message is set.

For dates beyond three days, the system switches to Enhanced Historical Pattern Estimation. It extracts the target month and day, consults long-term (30-year) meteorological data, and determines the relevant monsoon type and season. Using this information, it generates a consistent random seed to simulate variation and calculates an adjusted probability of rain. Based on this probability and seasonal context, the system classifies the condition: Heavy Rain/Thunderstorms, Scattered Showers, or Light Rain if rain is predicted, or otherwise Sunny/Hazy (dry season) or Partly/Mostly Cloudy (normal conditions). The source is set as "historical pattern enhanced," and an extended forecast disclaimer is displayed to manage user expectations.

Finally, the system classifies the overall weather result. If rainy conditions are detected and a rain alert has not been shown before, an alert is displayed and flagged to avoid duplication. The classified weather is then used to apply weather-based attraction filtering, ensuring that unsuitable outdoor activities are avoided in rainy conditions. The final weather condition result is then stored and returned to support itinerary planning.

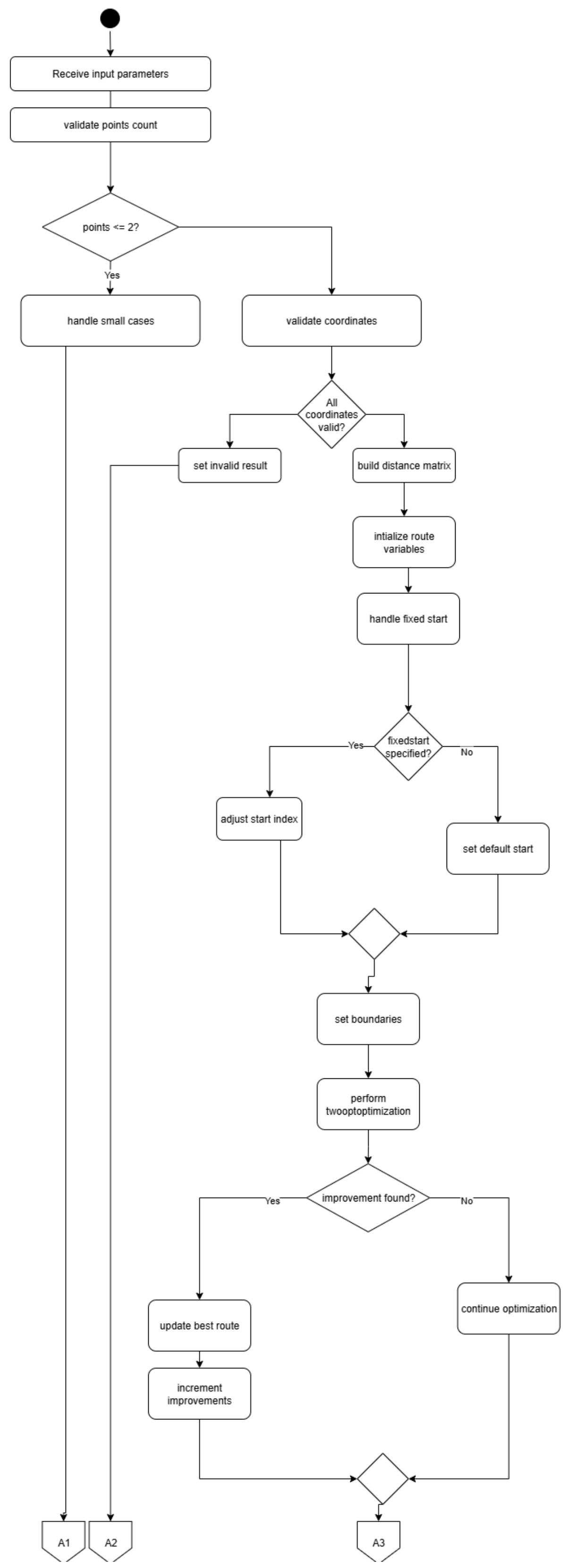
Use Case Description

Field	**Description**
Use Case Name	Weather Forecast
Actor	User (Traveler)
Description	The system provides weather forecasts for the user's trip dates. It integrates real-time WeatherAPI data for short-term forecasts (≤ 3 days) and enhanced historical pattern estimation for longer-term forecasts (> 3 days). Results are used to filter attractions and ensure weather-aware trip planning.
Trigger	User requests weather forecast for a planned trip.
Preconditions	- User must specify travel date(s) and location.

	- WeatherAPI service must be accessible (for ≤ 3 days forecast). - Historical meteorological dataset must be available (for > 3 days forecast).
Postconditions	- Weather condition is classified (e.g., sunny, rainy, cloudy). - Rain alert is displayed if applicable. - Weather-based attraction filtering is applied to itinerary planning.
Main Flow	1. Receive user inputs and validate them. 2. Calculate forecast days ahead. 3. If ≤ 3 days, use WeatherAPI to fetch real-time forecast. 4. Extract condition text and set data source. 5. If > 3 days, use Enhanced Historical Pattern Estimation. 6. Lookup long-term climate data, determine season/monsoon. 7. Generate rain probability and classify conditions. 8. Show extended forecast disclaimer (for historical-based results). 9. Classify weather and check if rain alert should be displayed. 10. Apply weather-based filtering to attractions. 11. Set and return weather condition result.
Alternative Flow	- 3a. WeatherAPI request fails → set error message .- 7a. If adjusted rain probability ≤ 30% → classify as “Light Rain” or “Partly Cloudy.” - 9a. If rain alert already shown → skip showing again.
Exceptions	- Invalid user inputs (missing/incorrect date/location). - API timeout or unresponsive. - Missing or corrupted historical dataset.

Table 3.3.3.6 Use Case Description – Weather Forecast

3.3.3.7 – Route Optimization



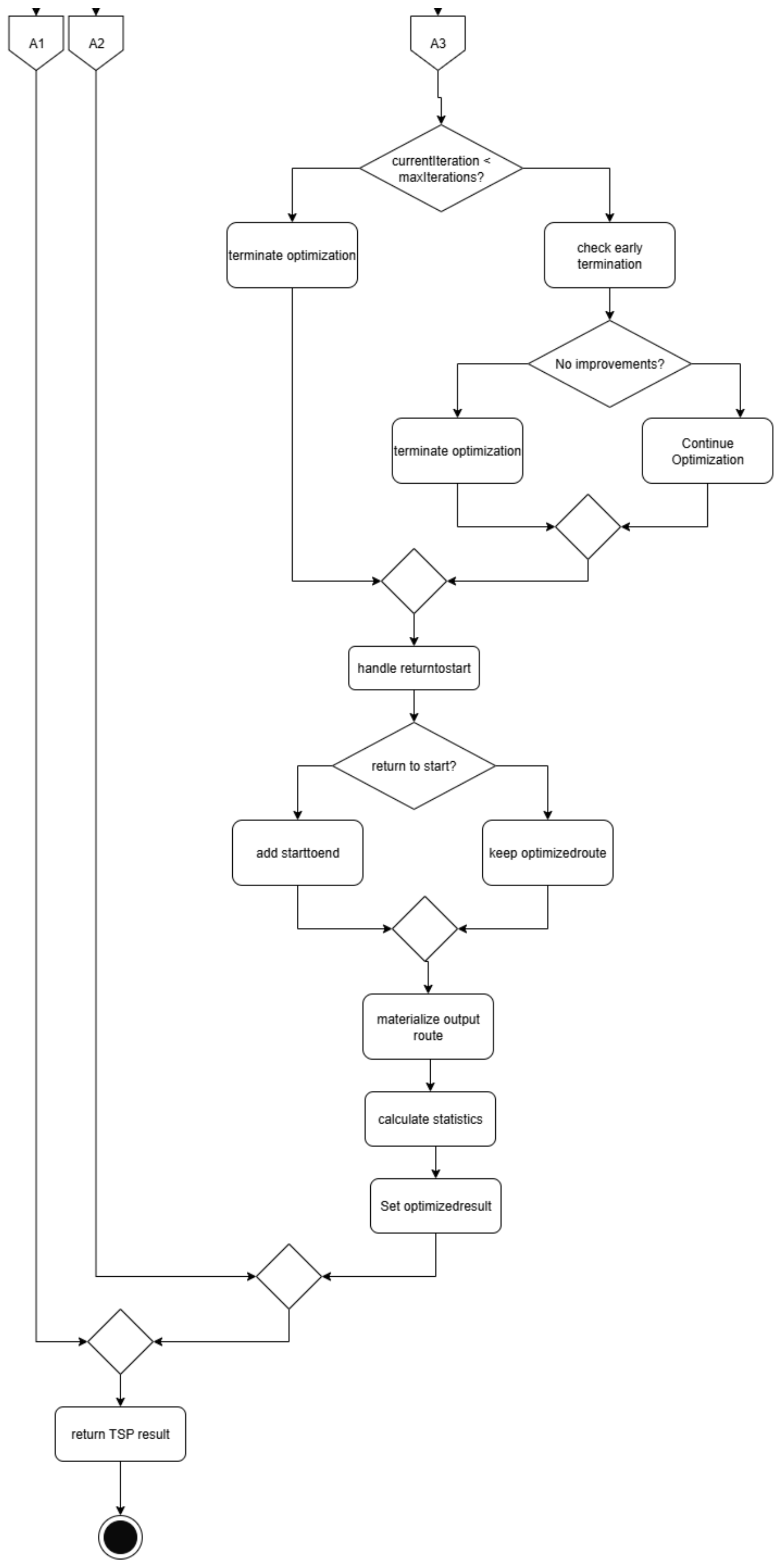


Figure 3.3.3.7 Activity Diagram – Route Optimization

The activity diagram represents the process of generating an optimized travel route using a Travelling Salesman Problem (TSP)-inspired approach. The workflow begins with receiving input parameters such as selected attractions and validating the number of points. If the number of points is two or fewer, the system handles these small cases directly. Otherwise, it proceeds to validate coordinates. If any coordinates are invalid, the result is set as invalid; if all are valid, the system builds a distance matrix and initializes route variables.

The algorithm then determines the starting point. If a fixed start is specified, the index is adjusted accordingly; otherwise, a default start is assigned. Boundaries are set, and a **2-opt optimization algorithm** is performed to iteratively refine the route. Whenever an improvement is found, the best route is updated, and the improvement counter is incremented. If no improvement is found, the algorithm continues optimization until early termination conditions or maximum iterations are reached.

After optimization ends, the system checks whether the route should return to the starting point. If so, the start is added to the end of the path; if not, the optimized route is kept as is. The system then materializes the final route, calculates relevant statistics (such as distance or duration), and sets the optimized result. Finally, the optimized TSP result is returned for use in itinerary planning.

This process ensures that the user receives an efficient route covering all selected attractions, while minimizing travel distance or time, and adapting flexibly to different trip configurations.

Use Case Description

Field	**Description**
Use Case Name	Route Optimization (TSP-style with 2-opt)
Primary Actor	System
Goal	Produce an efficient visiting order for selected places that minimizes total travel distance/time, with optional return to the start (round-trip).
Trigger	User selects places (and optionally a fixed start) and taps “Optimize Route” during trip planning.
Preconditions	1) At least one place selected. 2) Each place has valid coordinates. 3) Distance function available (e.g., Haversine).
Postconditions	1) Optimized route sequence is generated (or a valid small-case/invalid result is returned). 2) Route statistics (total distance/time) are computed and stored for the day’s itinerary.

Main Flow	1) System receives input parameters (places, fixed-start flag, round-trip flag, iteration limits). 2) Validate number of points and coordinates. 3) If points ≤ 2, handle small cases directly (A→B, and optionally B→A). 4) Build distance matrix for all valid points. 5) Initialize route (respect fixed start if provided; otherwise choose default start). 6) Set optimization boundaries (max iterations/early-stop). 7) Run 2-opt improvements iteratively over the route. 8) When an improvement is found, update best route and counters. 9) Stop when no improvement, early-stop triggers, or max iterations reached. 10) If round-trip requested, append start node at end; otherwise keep open path. 11) Materialize final route, compute statistics, and persist result.
Alternative Flows	A1 – Invalid Coordinates: On validation failure, return “invalid result” with error message and skip optimization.A2 – Fixed Start Not Found: If provided start ID not in list, fall back to default start and continue.A3 – Early Termination: If stagnation threshold reached (no improvements over N passes), terminate early and return best-so-far.
Exceptions	E1 – Empty Selection: No points provided → return empty route with notice.E2 – Duplicate or Coincident Points: Collapse duplicates before optimization; if all collapse to ≤2, use small-case handler.

Table 3.3.3.7 Use Case Description – Route Optimization

3.3.3.8 – Save Trip and Itinerary

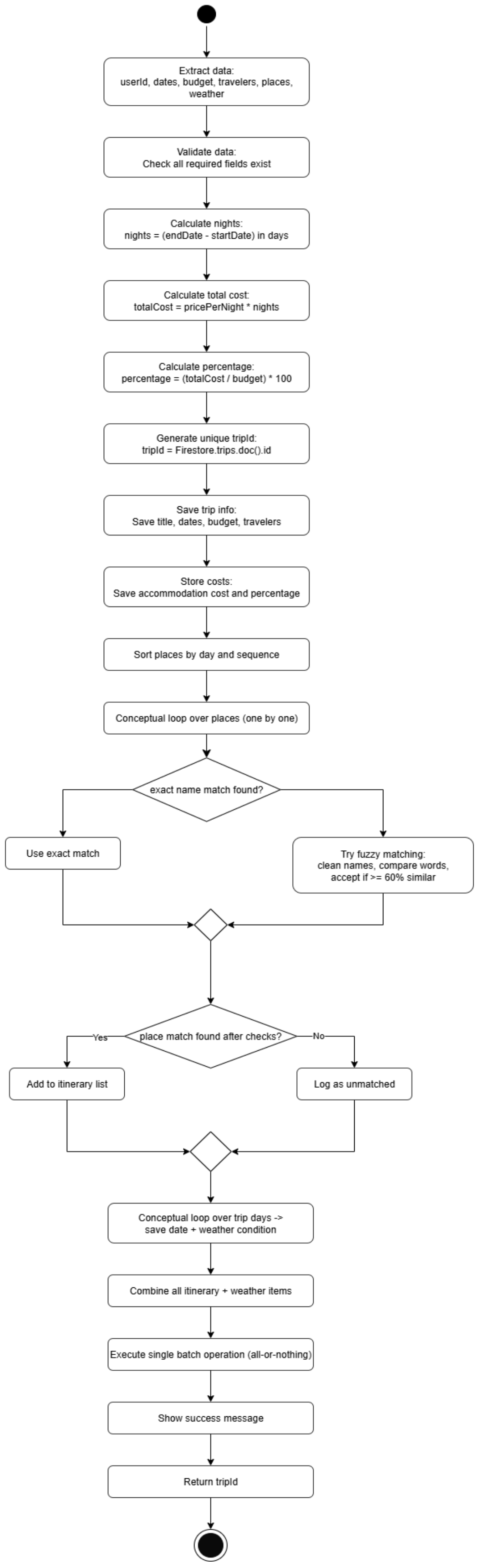

Extract data:
userId, dates, budget, travelers, places, weather
Validate data:
Check all required fields exist
Calculate nights:
nights = (endDate - startDate) in days
Calculate total cost:
totalCost = pricePerNight * nights
Calculate percentage:
percentage = (totalCost / budget) * 100
Generate unique tripId:
tripId = Firestore.trips.doc().id
Save trip info:
Save title, dates, budget, travelers
Store costs:
Save accommodation cost and percentage
Sort places by day and sequence
Conceptual loop over places (one by one)
exact name match found?
Use exact match
Try fuzzy matching:
clean names, compare words,
accept if >= 60% similar
place match found after checks?
Yes
No
Add to itinerary list
Log as unmatched
Conceptual loop over trip days ->
save date + weather condition
Combine all itinerary + weather items
Execute single batch operation (all-or-nothing)
Show success message
Return tripId

Figure 3.3.3.8 Activity Diagram – Save Trip and Itinerary

This system processes trip information by first extracting essential data including user ID, travel dates, budget, number of travelers, selected places, and weather information. The system validates this data to ensure all required fields are complete and properly formatted.

The system then performs key calculations including determining the trip duration in days by calculating the difference between end date and start date, computing the total cost by multiplying price per night by the number of nights, and calculating what percentage of the user's budget this represents using the formula (totalCost / budget) × 100.

After generating a unique trip ID using Firestore's document ID system, the system saves the basic trip information including title, dates, budget, and traveler count. It also stores cost details by saving accommodation expenses and the calculated budget percentage for tracking purposes.

The core itinerary generation process involves sorting all selected places by day and sequence to create a logical flow. The system then processes each place individually through a matching algorithm that first attempts exact name matching. If no exact match is found, it applies fuzzy matching by cleaning place names, comparing words, and accepting matches with 60% or higher similarity scores.

Successfully matched places are added to the itinerary list, while unmatched places are logged for review. The system then processes each trip day by saving date information along with corresponding weather conditions. Finally, it combines all itinerary data with weather information, executes a single batch operation to ensure data consistency, displays a success message, and returns the unique trip ID for future reference.

Field	**Description**
Use Case Name	Save Trip and Itinerary
Actor	User (Traveler)
Description	System processes and saves complete trip information, generates detailed itineraries with place matching, calculates costs, and stores all data with weather information in a single transaction
Preconditions	- User must have completed trip planning - Trip data (dates, budget, places, weather) must be available - Firestore database must be accessible - Place matching algorithms must be operational
Postconditions	- Trip is saved with unique ID - Complete itinerary is generated and stored - Cost calculations and budget analysis are completed

	- Weather data is integrated with daily plans - Success confirmation is provided to user
Main Flow	1. System extracts trip data (userID, dates, budget, travelers, places, weather) 2. System validates all required fields are complete 3. System calculates trip duration and total costs 4. System calculates budget percentage utilization 5. System generates unique trip ID 6. System saves basic trip information and cost details 7. System sorts places by day and sequence 8. **Place Matching Process**: System attempts exact name matching first 9. **If no exact match**: System applies fuzzy matching with 60% similarity threshold 10. System adds matched places to itinerary, logs unmatched item 11. System processes daily schedules with weather data 12. System combines all itinerary and weather information 13. System executes batch save operation 14. System displays success message and returns trip ID
Alternative Flows	**Alt 1**: Data validation fails - System prompts for missing information **Alt 2**: Place matching fails - System logs unmatched places and continues **Alt 3**: Database save fails - System retries operation or reports error **Alt 4**: Budget exceeds 100% - System flags budget warning but continues
Exceptions	- Missing required trip data - Invalid date ranges or budget values - Database connectivity issues - Place matching algorithm failures - Batch operation transaction failures

Table 3.3.3.8 Use Case Description – Save Trip and Itinerary

3.3.3.9 – Edit Itinerary

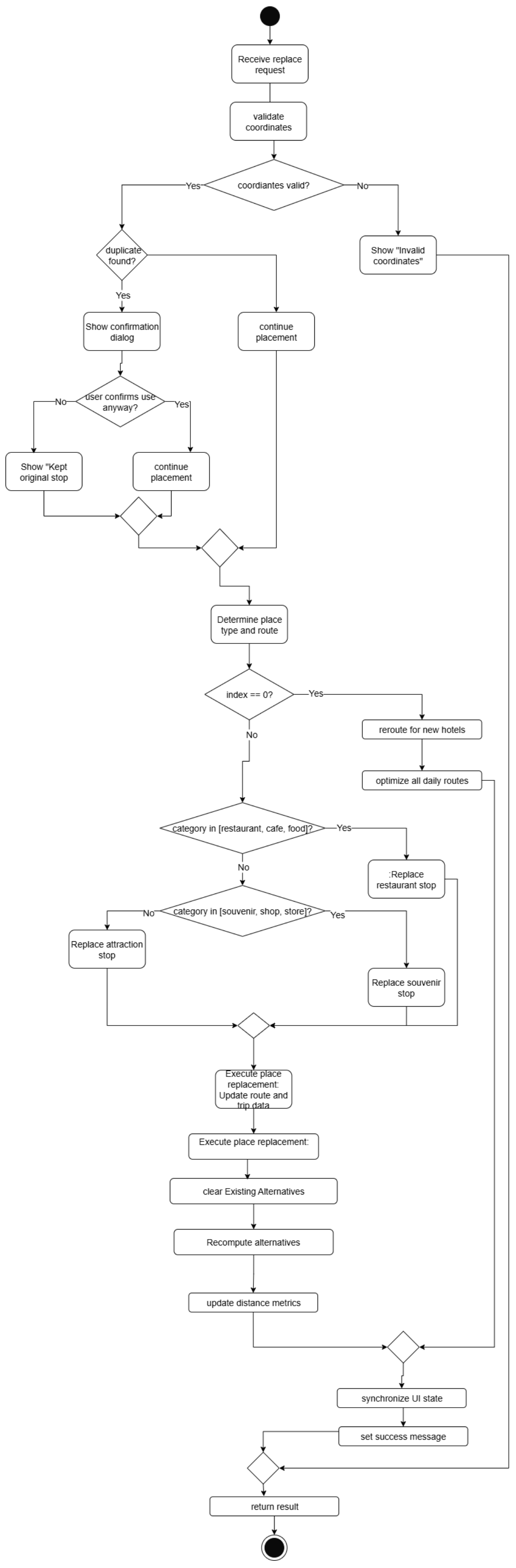


Figure 3.3.3.9 Activity Diagram – Edit Itinerary

This system manages itinerary modifications by first validating and preprocessing coordinates. The process begins by checking if the coordinates list has more than 2 points - if not, it shows "Invalid coordinates" and stops. For valid coordinate sets, the system detects duplicates using two methods: name comparison by normalizing strings, and location comparison by calculating distances with the formula: distance = $6371 \times 2 \times \text{asin}(\sqrt{a})$ where $a = \sin^2(\text{lat}) + \cos(\text{lat1}) \times \cos(\text{lat2}) \times \sin^2(\text{lon})$, setting duplicate threshold at 500 meters.

The system presents duplicate detection results to the user and asks for confirmation to proceed. If the user declines, it shows "Keep original stops" and exits. Upon confirmation, the system determines the place type and route optimization strategy.

If the place index is 0 (indicating a hotel), the system initiates hotel replacement by rerouting for the new hotel, updating all daily routes to start from the new location, and optimizing using nearest neighbor algorithms to calculate total distance as the sum of distances between consecutive points.

For non-hotel places, the system categorizes them as restaurants (cafe, food) or attractions (souvenir, shop, tourist sites). Based on the category, it replaces the restaurant stop or attraction stop accordingly, then replaces the souvenir shop if needed.

After any replacement, the system executes place replacement operations by updating daily point arrays, modifying place names and symbols, and refreshing formatted place names. It then clears and recomputes alternative place suggestions by searching within 2km radius for similar place types and filtering by category matching algorithms.

The system concludes by updating distance and route metrics through calculating total day distance as the sum of consecutive point distances, synchronizing the UI state by refreshing map markers and itinerary display, showing success confirmation, and returning "Place Successfully Replaced" status.

Use Case description

Field	**Description**
Use Case Name	Edit Itinerary
Actor	User (Traveler)
Description	System allows users to modify their travel itinerary by replacing hotels, restaurants, or attractions while maintaining route optimization and providing alternative suggestions
Preconditions	- User must have an existing itinerary - Coordinate data must be available

	- Place database must be accessible - Route optimization algorithms must be operational
Postconditions	- Selected place is successfully replaced in itinerary - Route distances are recalculated and optimized - Alternative place suggestions are updated - UI displays reflect changes - Success confirmation is provided
Main Flow	1. System validates coordinates list (must have >2 points) 2. System detects duplicates using name comparison and 500m distance threshold 3. System shows duplicate results and requests user confirmation 4. System determines place type and route optimization strategy 5. **If Hotel (index=0)**: System reroutes for new hotel and optimizes all daily routes 6. **If Restaurant**: System replaces restaurant stop and updates category 7. **If Attraction**: System replaces attraction/souvenir shop accordingly 8. System executes place replacement by updating point arrays and place names 9. System clears and recomputes alternative suggestions within 2km radius 10. System updates distance calculations and route metrics 11. System synchronizes UI state and refreshes displays 12. System shows success confirmation
Alternative Flows	**Alt 1**: Invalid coordinates (<2 points) - System shows error and stops **Alt 2**: User declines duplicate confirmation - System keeps original stops **Alt 3**: No suitable replacements found - System suggests expanding search criteria **Alt 4**: Route optimization fails - System maintains original route structure
Exceptions	- Insufficient coordinate data provided - Place database unavailable - Route optimization algorithm failures - UI synchronization errors - Distance calculation computation errors

Table 3.3.3.9 Use Case Description – Edit Itinerary

3.3.3.10 – Book Accommodations

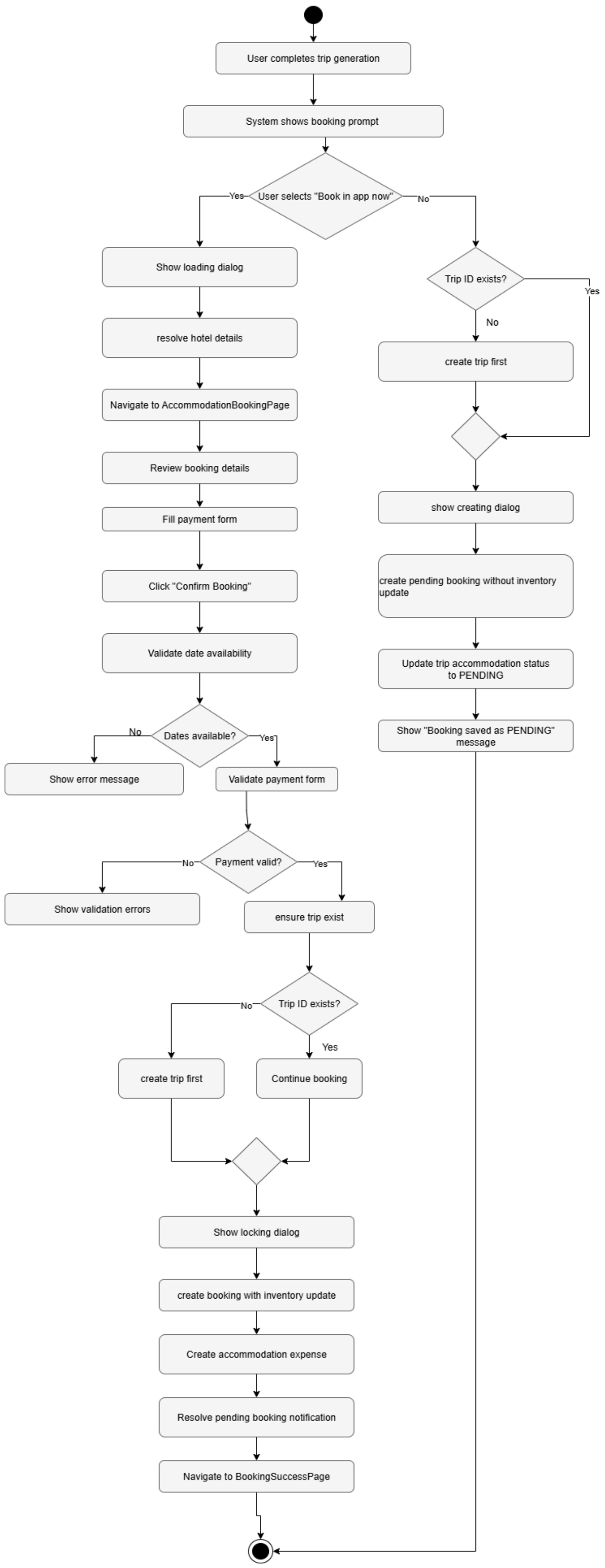

User completes trip generation
System shows booking prompt
User selects "Book in app now"
Yes
No
Show loading dialog
resolve hotel details
Navigate to AccommodationBookingPage
Review booking details
Fill payment form
Click "Confirm Booking"
Validate date availability
Dates available?
No
Yes
Show error message
Validate payment form
Payment valid?
No
Yes
Show validation errors
ensure trip exist
Trip ID exists?
No
Yes
create trip first
Continue booking
Show locking dialog
create booking with inventory update
Create accommodation expense
Resolve pending booking notification
Navigate to BookingSuccessPage
Trip ID exists?
Yes
No
create trip first
show creating dialog
create pending booking without inventory update
Update trip accommodation status to PENDING
Show "Booking saved as PENDING" message

Figure 3.3.3.10 Activity Diagram – Book Accommodations

This accommodation booking system follows a structured flow that begins after trip generation completion. The system presents users with a booking prompt bottom sheet offering two distinct pathways.

Initial Decision Point: Users choose between "Book in app now" for immediate booking or "Skip for now" for deferred booking.

When users select immediate booking, the system initiates a loading dialog and executes ensureResolvedHotel() to retrieve comprehensive hotel data from Firebase. This function systematically attempts hotel photo resolution through placeDocPath lookup, collectionGroup queries, and fuzzy matching fallback methods. Upon successful data retrieval, the system navigates to AccommodationBookingPage where users review hotel galleries, dates, room selections, and pricing.

Users complete payment forms with cardholder information, card details, and billing email, then proceed to "Confirm Booking". The system validates date availability through _dateRangeAvailable() and payment information using form validators. If validation fails, appropriate error messages display and the process halts. For successful validation, the system checks trip existence and calls saveTripCallback if needed.

The core booking execution involves _createBookingWithInventory() which generates confirmation codes, creates Firebase booking documents, and triggers _decreaseHotelInventory() to update room availability through Firestore transactions. The system automatically generates accommodation expenses via _upsertAccommodationExpenseFromBooking(), resolves pending notifications, and navigates to BookingSuccessPage for confirmation display.

Users selecting "Skip for now" trigger trip creation if necessary, followed by _createPendingBooking() execution which creates PENDING status bookings with essential data storage. The system updates trip accommodation status to PENDING, emits booking_pending notifications, and navigates to MainScreen with confirmation messaging.

The system handles notifications through _emitBookingPendingNotification() for deferred bookings and _resolvePendingBookingNotification() for confirmed bookings, creating appropriate notification documents with booking details and action triggers.

Use Case Description

Field	Description
Use Case Name	Book Accommodation with Firebase Integration
Actor	User (Traveler)
Description	System enables immediate or deferred accommodation booking through structured flow pathways, using Firebase for data management, inventory control, expense tracking, and notification handling
Preconditions	- User must have completed trip generation - Firebase database must be accessible - Hotel inventory data must be available in Firebase - Payment validation system must be operational - Hotel data resolution functions must be available
Postconditions	- Booking is created with CONFIRMED or PENDING status in Firebase - Hotel inventory is updated through Firestore transactions (confirmed bookings) - Accommodation expenses are automatically generated and stored - Notification documents are created and managed appropriately - Trip status is updated to reflect booking state
Main Flow	1. System displays booking prompt bottom sheet after trip generation 2. **Immediate Booking Flow**: User selects "Book in app now" 3. System executes ensureResolvedHotel() using placeDocPath, collectionGroup queries, fuzzy matching 4. Navigate to AccommodationBookingPage for review and payment form completion 5. System validates availability (_dateRangeAvailable) and payment forms 6. System creates trip via saveTripCallback if needed 7. Execute _createBookingWithInventory() to generate confirmation and update inventory 8. Run _decreaseHotelInventory() through Firestore transactions 9. Create expenses via _upsertAccommodationExpenseFromBooking(10. **Deferred Booking Flow**: User selects "Skip for now" 11. Execute _createPendingBooking() with PENDING status 12. Emit notifications (_emitBookingPendingNotification or _resolvePendingBookingNotification) 13. Navigate to appropriate confirmation or pending status pages

Alternative Flows	**Alt 1**: Hotel resolution fails - System uses fallback data without photos and continues **Alt 2**: Date unavailability - System shows error message and halts booking process **Alt 3**: Payment validation failure - System displays validation errors and requests correction **Alt 4**: Inventory update failure - System logs error but continues booking process **Alt 5**: Expense creation failure - System generates warning but maintains booking confirmation
Exceptions	- Firebase connectivity issues during data retrieval - Firestore transaction failures during inventory updates - Payment validation system unavailable - Hotel data resolution complete failure - Notification system failures - Trip creation callback errors

Table 3.3.3.10 Use Case Description – Book Accommodations

.

3.3.3.11 – Alert user when budget nearly exceeds

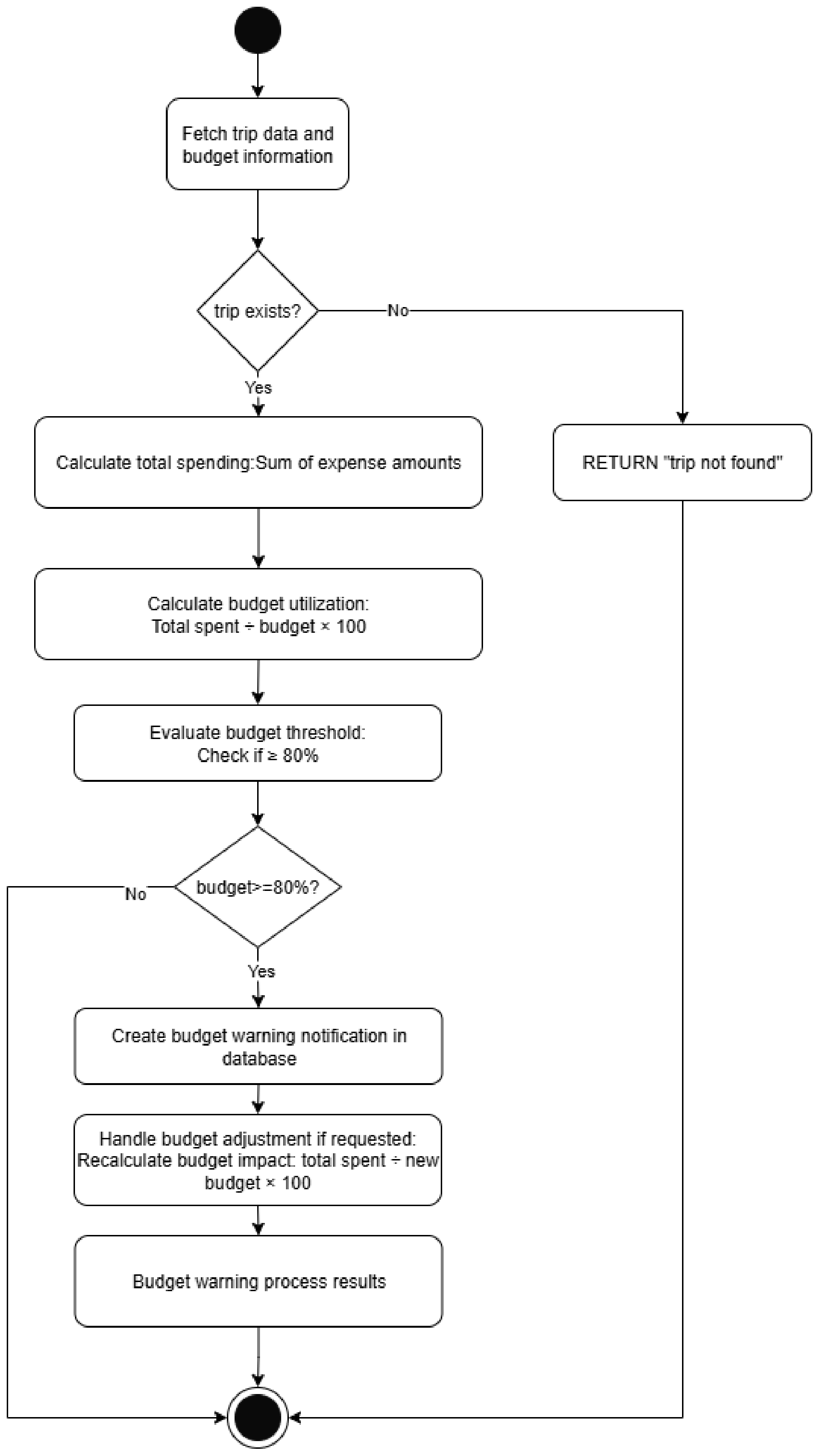

Fetch trip data and budget information
trip exists?
No
Yes
Calculate total spending:Sum of expense amounts
RETURN "trip not found"
Calculate budget utilization: Total spent ÷ budget × 100
Evaluate budget threshold: Check if ≥ 80%
budget>=80%?
No
Yes
Create budget warning notification in database
Handle budget adjustment if requested: Recalculate budget impact: total spent ÷ new budget × 100
Budget warning process results

Figure 3.3.3.11 Activity Diagram – Alert user when budget nearly exceeds

This budget monitoring system operates through a straightforward flow designed to help users manage their travel expenses. The process begins by fetching trip data and budget information from the database. The system first validates that the trip exists - if no trip is found, it logs "Trip not found" and terminates the process.

For valid trips, the system calculates total spending by summing all expense amounts recorded for the trip. It then determines budget utilization using the formula: (Total spent ÷ budget) × 100 to get a percentage. The core logic evaluates whether this budget utilization has reached or exceeded the 80% threshold.

If spending remains below 80% of the budget, the system logs "Under control" and ends the monitoring process. However, when budget utilization reaches or exceeds 80%, the system triggers the warning mechanism by creating a budget warning notification in the database to alert the user about their approaching budget limit.

The system also includes functionality to handle budget adjustments if users request changes to their budget. When budget modifications occur, it recalculates the budget impact using the new budget amount with the same formula: (total spent ÷ new budget) × 100, then processes the updated budget warning results accordingly.

Use case description

Field	**Description**
Use Case Name	Alert user when budget nearly exceeds
Actor	System (Automated), User (Traveler)
Description	System automatically monitors trip expenses and generates alerts when spending approaches 80% of the allocated budget, with support for budget adjustments
Preconditions	- Trip must exist in the database - Budget amount must be set for the trip - Expense data must be available - Notification system must be operational
Postconditions	- Budget utilization is calculated and monitored - Warning notification is created when threshold exceeded - Budget adjustment impacts are recalculated if needed - Appropriate logging is performed for tracking
Main Flow	1. System fetches trip data and budget information from database 2. System validates trip existence 3. System calculates total spending by summing all expense amounts 4. System calculates budget utilization: (Total spent ÷ budget) × 100 5. System evaluates if budget utilization ≥ 80% 6. **If under 80%**: System logs "Under control" and terminates 7. **If ≥ 80%**: System creates budget warning notification in database 8. **If budget adjustment requested**: System recalculates with new budget

	9. System processes budget warning results and completes monitoring
Alternative Flows	**Alt 1**: Trip not found - System logs error and returns without processing **Alt 2**: Budget under control - System logs status and terminates monitoring **Alt 3**: Budget adjustment - System recalculates utilization with new budget amount
Exceptions	- Trip data unavailable or corrupted - Missing budget information - Expense calculation errors - Database connectivity issues - Notification system failures

Table 3.3.3.11 Use Case Description – Alert user when budget nearly exceeds

3.3.3.12 – Split Expenses

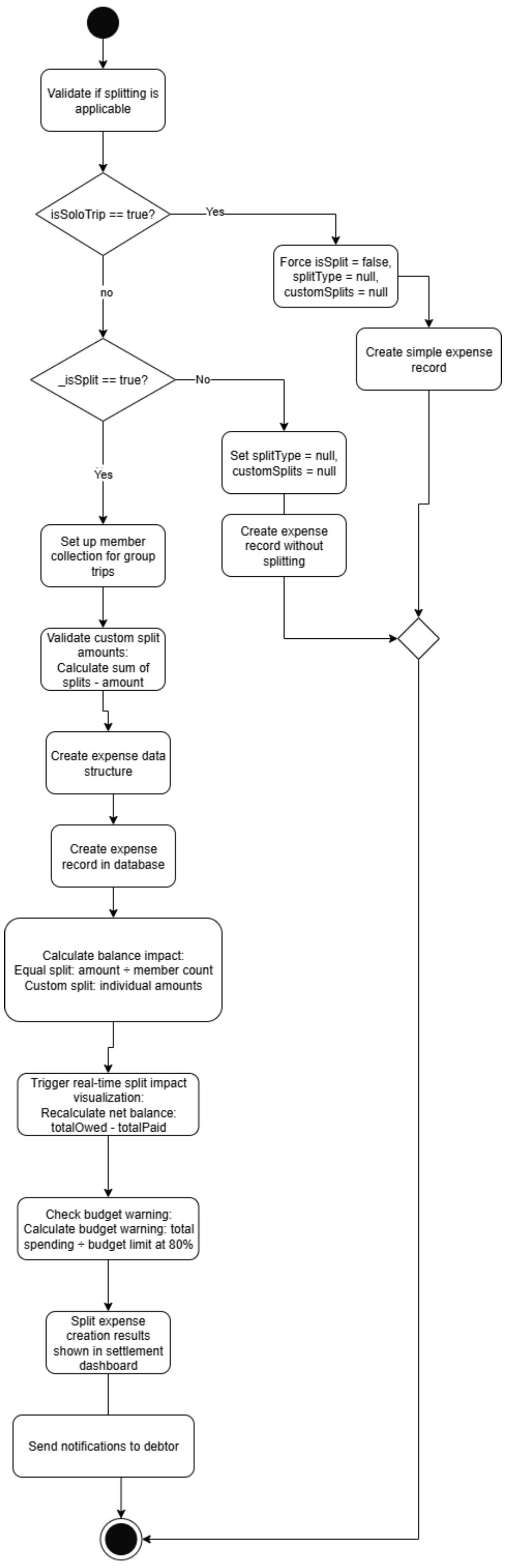

Validate if splitting is applicable
isSoloTrip == true?
Yes
Force isSplit = false, splitType = null, customSplits = null
Create simple expense record
no
_isSplit == true?
No
Set splitType = null, customSplits = null
Create expense record without splitting
Yes
Set up member collection for group trips
Validate custom split amounts: Calculate sum of splits - amount
Create expense data structure
Create expense record in database
Calculate balance impact: Equal split: amount ÷ member count Custom split: individual amounts
Trigger real-time split impact visualization: Recalculate net balance: totalOwed - totalPaid
Check budget warning: Calculate budget warning: total spending ÷ budget limit at 80%
Split expense creation results shown in settlement dashboard
Send notifications to debtor

Figure 3.3.3.12 Activity Diagram – Split Expenses

This expense splitting system operates through a validation-driven flow that determines how costs should be allocated among group members. The process begins by validating if splitting is applicable to the current expense scenario.

The system first checks if this is a solo trip - if isSoloTrip is true, it forces isSplit to false, sets splitType and customSplits to null, creates a simple expense record without any splitting functionality, and terminates the process. For group trips, the system continues to the next validation step.

The second validation checks the _isSplit flag. If splitting is disabled (_isSplit is false), the system sets splitType and customSplits to null, creates an expense record without splitting calculations, and stops processing. When splitting is enabled, the system proceeds to the splitting logic.

For valid splitting scenarios, the system establishes the member collection for group trips and validates custom split amounts by calculating the sum of individual splits against the total expense amount to ensure accuracy. The system then creates the expense data structure and stores the expense record in the database.

The core calculation process determines balance impact using two methods: for equal splits, it divides the total amount by member count (amount ÷ member count), while for custom splits, it uses individually specified amounts for each member. The system triggers real-time split impact visualization by recalculating net balances using the formula: totalOwed - totalPaid for each member.

Finally, the system performs budget monitoring by checking if total spending approaches the budget warning threshold at 80% utilization (total spending ÷ budget limit), then completes the split expense creation process with all calculated results.

Use Case Description

Field	**Description**
Use Case Name	Cost Split
Actor	User (Group Member), Travel Management System
Description	System manages expense splitting among group members with validation logic, supporting both equal and custom split allocations while monitoring budget impact
Preconditions	- Group trip must be established (if applicable) - Member collection must be available for group trips - Expense amount and details must be provided

	- Split preferences must be configured - Database must be accessible for expense storage
Postconditions	- Expense record is created with appropriate splitting logic - Member balances are updated based on split calculations - Real-time split visualization is triggered - Budget warning check is performed - Split impact is calculated and stored
Main Flow	1. System validates if expense splitting is applicable 2. **Solo Trip Check**: If solo trip, force disable splitting and create simple expense 3. **Split Flag Check**: If splitting disabled, create expense without split calculations 4. System sets up member collection for group expense splitting 5. System validates custom split amounts against total expense 6. System creates expense data structure and database record 7. **Equal Split**: Calculate amount ÷ member count for each member 8. **Custom Split**: Use individually specified amounts per member 9. System triggers real-time balance visualization (totalOwed - totalPaid) 10. System checks budget warning at 80% threshold 11. System completes split expense creation with all results
Alternative Flows	**Alt 1**: Solo trip detected - System disables splitting and creates simple expense **Alt 2**: Splitting disabled - System creates expense without split calculations **Alt 3**: Custom split validation fails - System prompts for correction **Alt 4**: Budget warning triggered - System alerts about spending threshold
Exceptions	- Invalid member collection for group trips - Custom split amounts don't match total expense - Database connectivity issues during expense creation - Balance calculation errors - Budget monitoring system failures

Table 3.3.3.12 Use Case Description – Split Expenses

3.3.3.13 – Make Payment

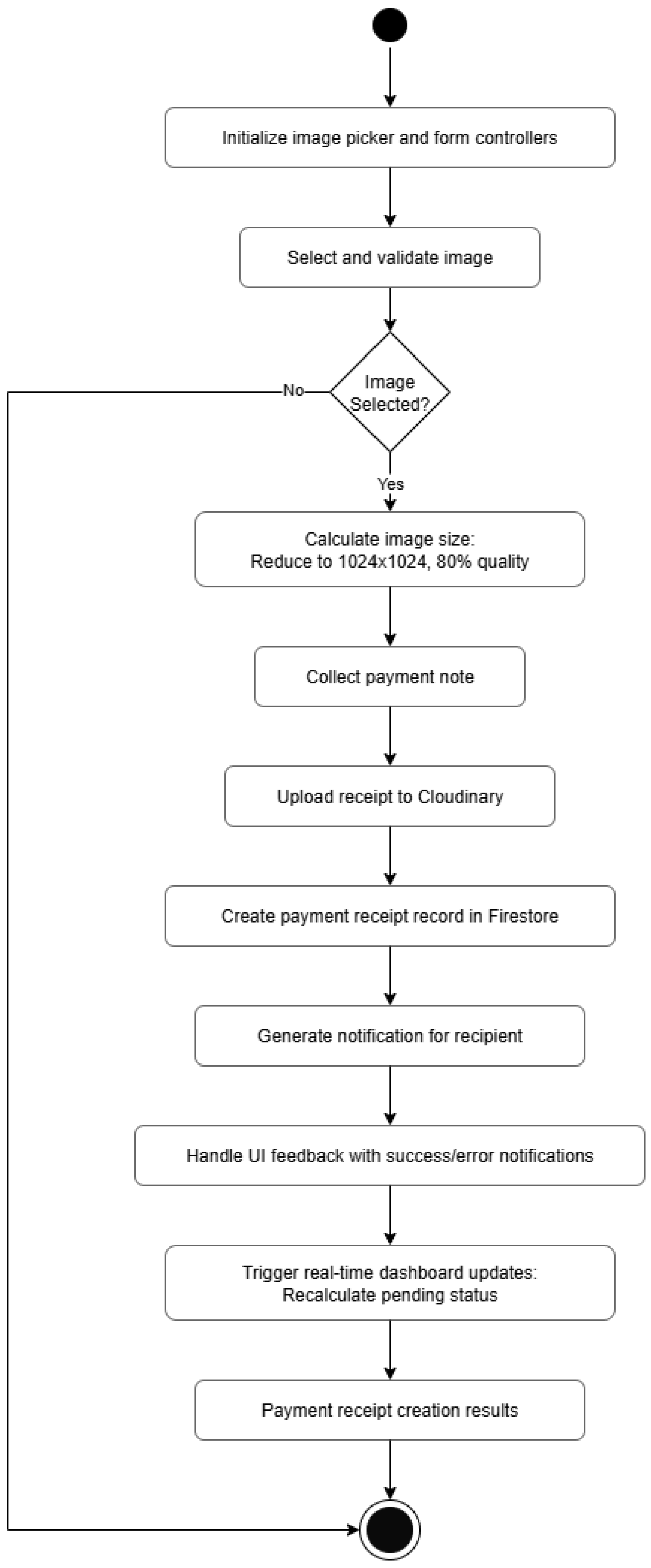

Initialize image picker and form controllers
Select and validate image
Image Selected?
No
Yes
Calculate image size: Reduce to 1024x1024, 80% quality
Collect payment note
Upload receipt to Cloudinary
Create payment receipt record in Firestore
Generate notification for recipient
Handle UI feedback with success/error notifications
Trigger real-time dashboard updates: Recalculate pending status
Payment receipt creation results

Figure 3.3.3.13 Activity Diagram – Make Payment

This payment processing system manages receipt-based payments through a streamlined digital workflow. The process begins by initializing image picker controllers and form management components to prepare for receipt capture and data entry.

The system prompts users to select and validate a payment receipt image. If no image is selected, the system maintains the current state without making changes and terminates the process. When an image is successfully selected, the system proceeds with payment processing. The image optimization process calculates and reduces the image size to 1024x1024 pixels with 80% quality compression to ensure efficient storage and upload performance. Users then provide payment notes or descriptions to accompany the receipt documentation.

The system uploads the processed receipt image to Cloudinary for cloud storage and creates a corresponding payment receipt record in Firestore database for data persistence. Upon successful record creation, the system generates notifications for the payment recipient to inform them about the transaction.

The system handles user interface feedback by displaying appropriate success or error notifications based on the processing outcome. Finally, it triggers real-time dashboard updates by recalculating pending payment status and updating all relevant displays to reflect the new payment information.

Use Case Description

Field	**Description**
Use Case Name	Make Payment
Actor	User (Payer), Payment Recipient
Description	System processes payments through receipt image upload, handling image optimization, cloud storage, database recording, and real-time notification updates
Preconditions	- User must have payment obligation or expense to settle - Image picker functionality must be available - Cloudinary cloud storage must be accessible - Firestore database must be operational - Notification system must be functional
Postconditions	- Payment receipt is uploaded and stored in Cloudinary - Payment record is created in Firestore database - Recipient receives payment notification - Dashboard displays are updated with new payment status -Pending payment calculations are refreshed

Main Flow	1. System initializes image picker and form controllers 2. User selects and validates payment receipt image 3. **If no image selected**: System maintains current state and terminates 4. **If image selected**: System calculates and reduces image to 1024x1024, 80% quality 5. User provides payment notes and description 6. System uploads optimized receipt to Cloudinary cloud storage 7. System creates payment receipt record in Firestore database 8. System generates notification for payment recipient 9. System handles UI feedback with success/error notifications 10. System triggers real-time dashboard updates and recalculates pending status
Alternative Flows	**Alt 1**: No image selected - System maintains current state without processing **Alt 2**: Image upload fails - System shows error notification and allows retry **Alt 3**: Database creation fails - System logs error and requests user retry **Alt 4**: Notification delivery fails - System records payment but alerts about notification issue
Exceptions	- Image picker initialization failures - Invalid or corrupted image files - Cloudinary upload service unavailable - Firestore database connectivity issues - Notification system failures - Network connectivity problems during upload

Table 3.3.3.13 Use Case Description – Make Payment

3.3.3.14 – Settle Up Expenses

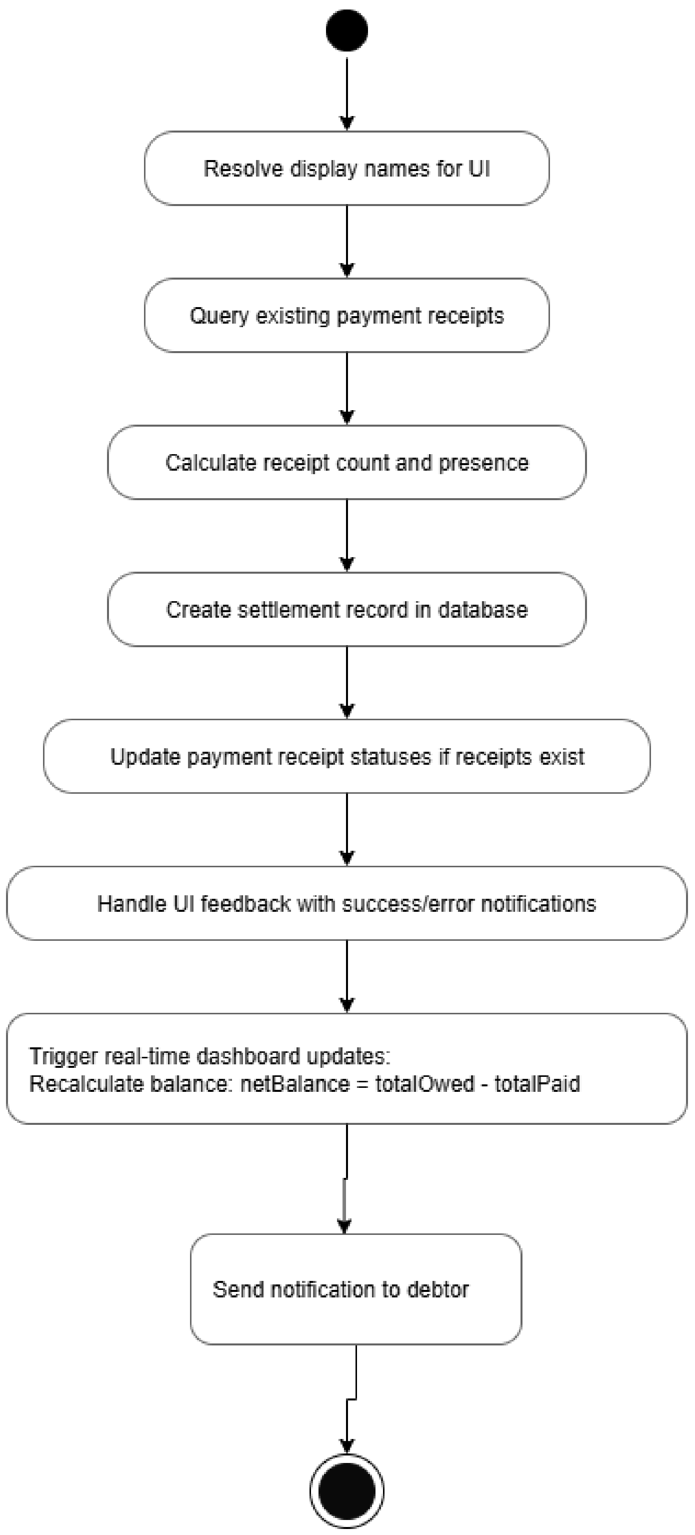

Resolve display names for UI
Query existing payment receipts
Calculate receipt count and presence
Create settlement record in database
Update payment receipt statuses if receipts exist
Handle UI feedback with success/error notifications
Trigger real-time dashboard updates:
Recalculate balance: netBalance = totalOwed - totalPaid
Send notification to debtor

Figure 3.3.3.14 Activity Diagram – Settle Up Expenses

This settlement system manages the final resolution of payment obligations between group members through a structured database-driven process. The system begins by resolving display names for user interface presentation to ensure clear identification of all parties involved in the settlement.

The process queries existing payment receipts from the database to gather comprehensive information about previous transactions and payment documentation. The system calculates receipt count and determines the presence of supporting documentation for the settlement process.

The core settlement operation involves creating a settlement record in the database to formally document the debt resolution between parties. If payment receipts exist from previous transactions, the system updates their statuses to reflect the completed settlement, ensuring all related documentation is properly linked and accounted for.

The system provides user interface feedback through success or error notifications to inform users about the settlement completion status. Finally, it triggers real-time dashboard updates by recalculating balances using the formula: netBalance = totalOwed - totalPaid, ensuring all displays reflect the updated financial status after settlement completion.

Use Case Description

Field	**Description**
Use Case Name	Settle Up Expenses
Actor	User (Group Member), Group Participants
Description	System finalizes outstanding payment obligations between group members by creating settlement records, updating receipt statuses, and recalculating balances
Preconditions	- Outstanding balances must exist between group members - User display names must be resolvable - Database must be accessible for settlement recording - Payment receipt data must be available (if applicable)
Postconditions	- Settlement record is created in database - Payment receipt statuses are updated (if receipts exist) - Net balances are recalculated and updated - Dashboard displays reflect current settlement status - Success/error notifications are provided to users
Main Flow	1. System resolves display names for UI presentation 2. System queries existing payment receipts from database 3. System calculates receipt count and determines receipt presence 4. System creates settlement record in database 5. **If receipts exist**: System updates payment receipt statuses

	6. System handles UI feedback with success/error notifications 7. System triggers real-time dashboard updates 8. System recalculates balance: netBalance = totalOwed – totalPaid 9. System completes settlement process with updated results
Alternative Flows	**Alt 1**: No existing receipts - System creates settlement without receipt status updates **Alt 2**: Database creation fails - System shows error notification and allows retry **Alt 3**: Receipt status update fails - System logs warning but completes settlement **Alt 4**: Balance calculation errors - System alerts about calculation issues
Exceptions	- Database connectivity issues during settlement creation - Display name resolution failures - Payment receipt query errors - Balance calculation computation errors - Dashboard update synchronization failures

Table 3.3.3.14 Use Case Description – Settle Up Expenses

3.3.3.15 – View Group Balances

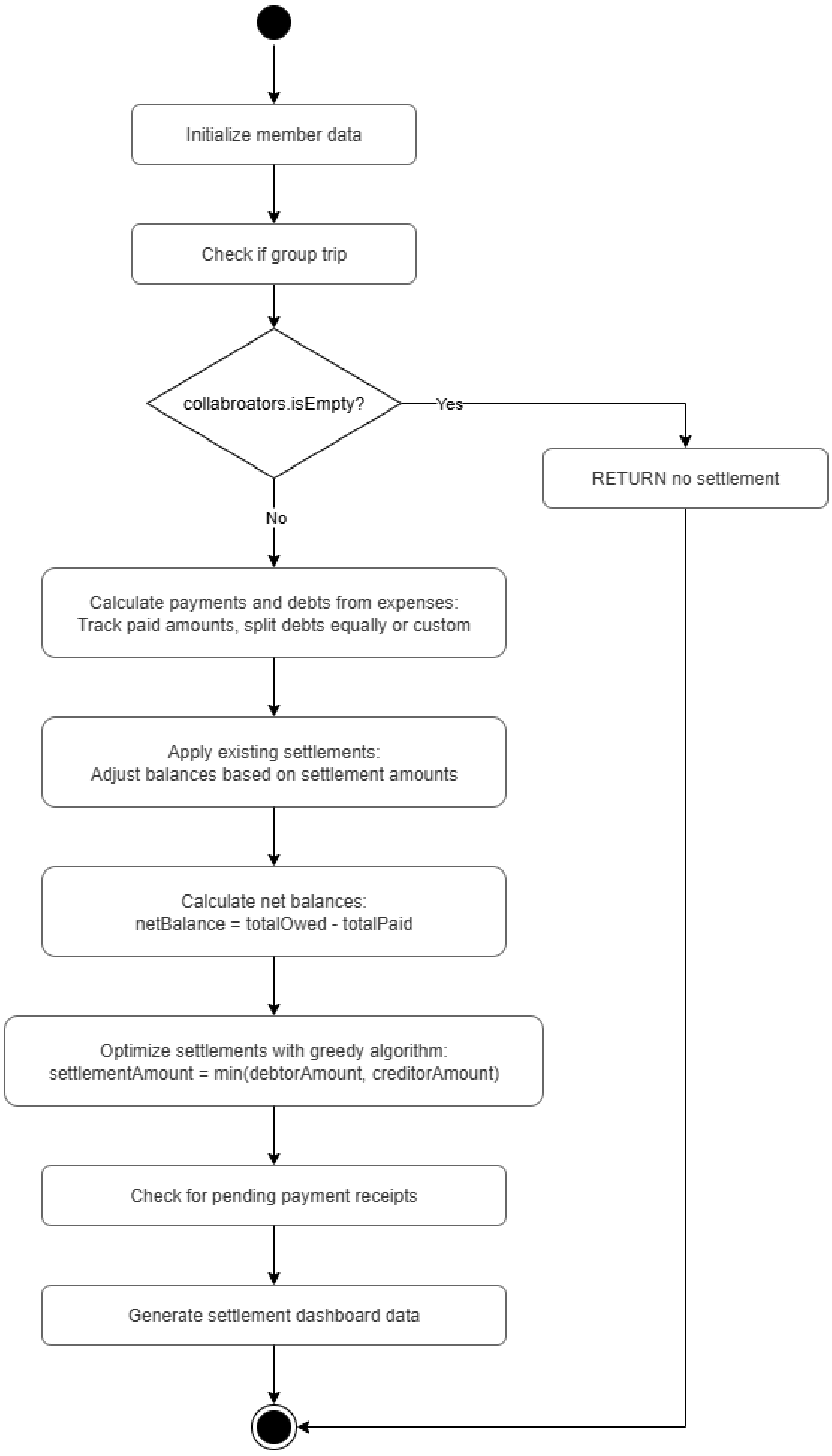

Initialize member data
Check if group trip
collabroators.isEmpty?
Yes
RETURN no settlement
No
Calculate payments and debts from expenses:
Track paid amounts, split debts equally or custom
Apply existing settlements:
Adjust balances based on settlement amounts
Calculate net balances:
netBalance = totalOwed - totalPaid
Optimize settlements with greedy algorithm:
settlementAmount = min(debtorAmount, creditorAmount)
Check for pending payment receipts
Generate settlement dashboard data

Figure 3.3.3.15 Activity Diagram – View Group Balances

This settlement dashboard system manages group financial balance visualization through a comprehensive calculation and optimization process. The system begins by initializing member data and checking if this is a group trip scenario. If the collaborators list is empty, indicating no group members exist, the system returns "no settlement" and terminates processing.

For valid group trips with collaborators, the system calculates payments and debts from recorded expenses by tracking paid amounts and applying either equal split distributions or custom split arrangements based on the expense configuration. The system then applies existing settlements by adjusting current balances based on previously recorded settlement amounts to ensure accurate financial positions.

The core balance calculation determines net balances for each member using the formula: netBalance = totalOwed - totalPaid, providing a clear picture of who owes money and who should receive payments. The system optimizes settlement recommendations using a greedy algorithm that calculates optimal payment amounts with: settlementAmount = min(debtorAmount, creditorAmount), ensuring efficient debt resolution with minimal transactions.

The system checks for pending payment receipts to incorporate recent payment submissions that may affect balance calculations. Finally, it generates comprehensive settlement dashboard data that presents all financial relationships, recommended settlements, and current balance status for group review and action.

Use Case Description

Field	**Description**
Use Case Name	View Group Balances
Actor	User (Group Member), Group Participants
Description	System displays comprehensive group financial dashboard showing member balances, debt relationships, and optimized settlement recommendations using algorithmic calculations
Preconditions	- Group trip must exist with collaborators - Expense data must be available - Settlement history must be accessible - Member data must be initialized - Payment receipt data must be queryable
Postconditions	- Complete settlement dashboard is generated and displayed - Net balances are calculated for all group members

	- Optimized settlement recommendations are provide - Pending payment receipts are incorporated - Financial relationships are clearly visualized
Main Flow	1. System initializes member data for group trip 2. System checks for collaborator existence (empty check) 3. **If no collaborators**: Return "no settlement" and terminate 4. **If collaborators exist**: Calculate payments and debts from expenses 5. System tracks paid amounts and applies split distributions (equal or custom) 6. System applies existing settlements to adjust current balances 7. System calculates net balances: netBalance = totalOwed – totalPaid 8. System optimizes settlements using greedy algorithm 9. System calculates: settlementAmount = min(debtorAmount, creditorAmount) 10. System checks for pending payment receipts 11. System generates comprehensive settlement dashboard data
Alternative Flows	**Alt 1**: No collaborators found - System returns no settlement status **Alt 2**: No expenses recorded - System shows balanced state **Alt 3**: All balances settled - System displays zero net balances **Alt 4**: Pending receipts affect calculations - System incorporates recent payments
Exceptions	- Member data initialization failures - Expense calculation errors - Settlement history retrieval issues - Balance calculation computation errors - Payment receipt query failures

Table 3.3.3.15 Use Case Description – View Group Balances

3.3.4 ERD

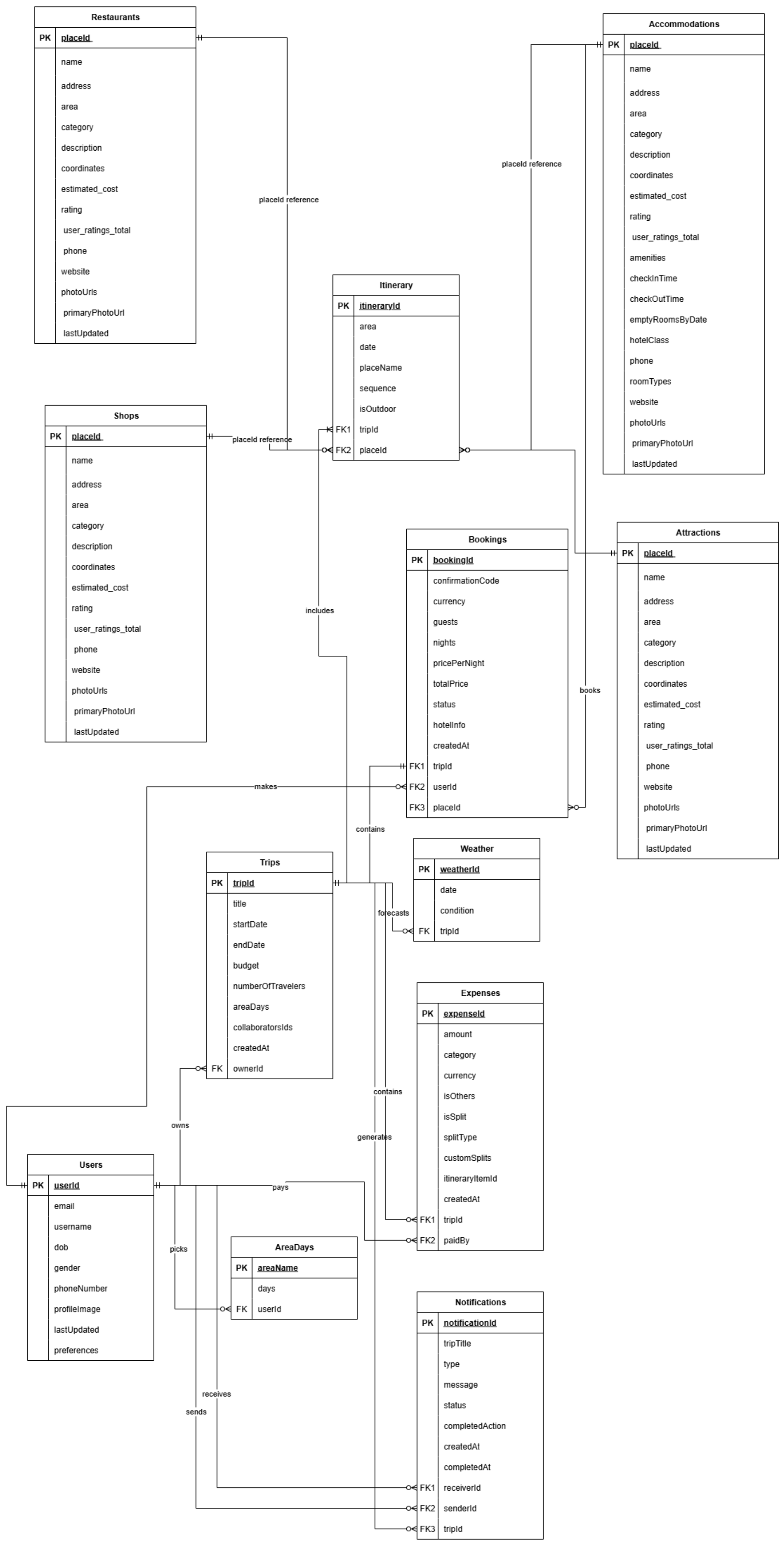
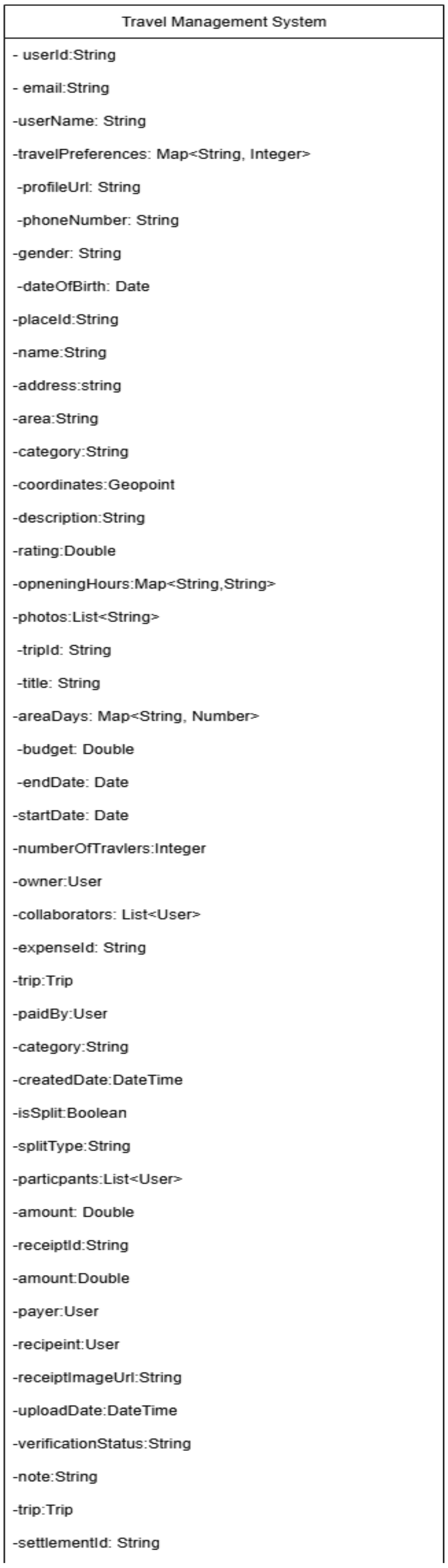
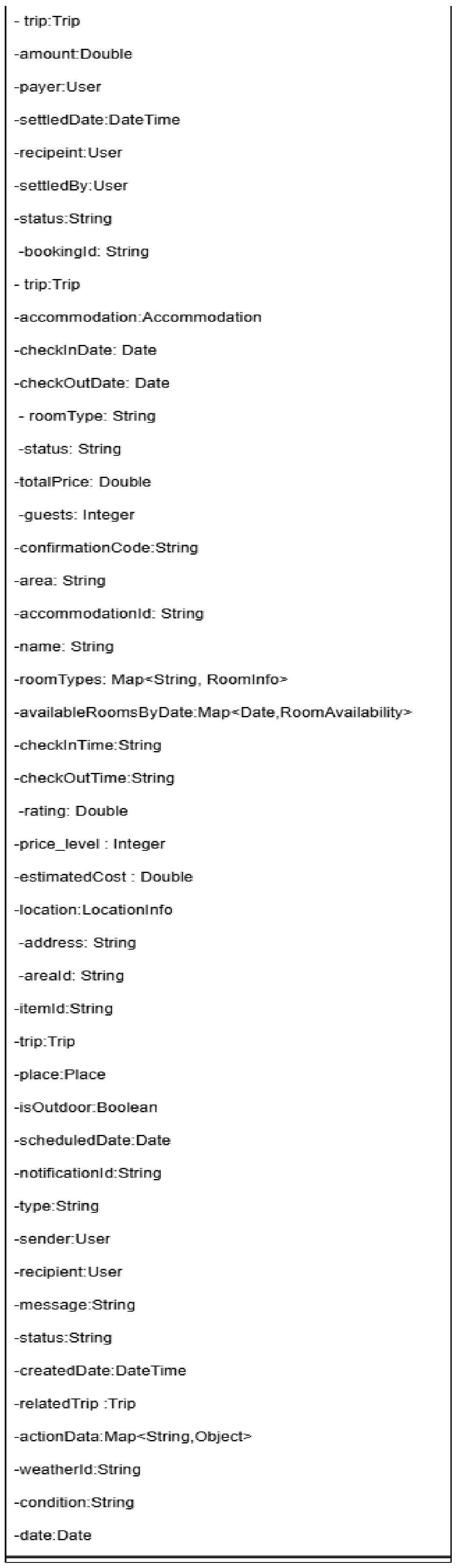


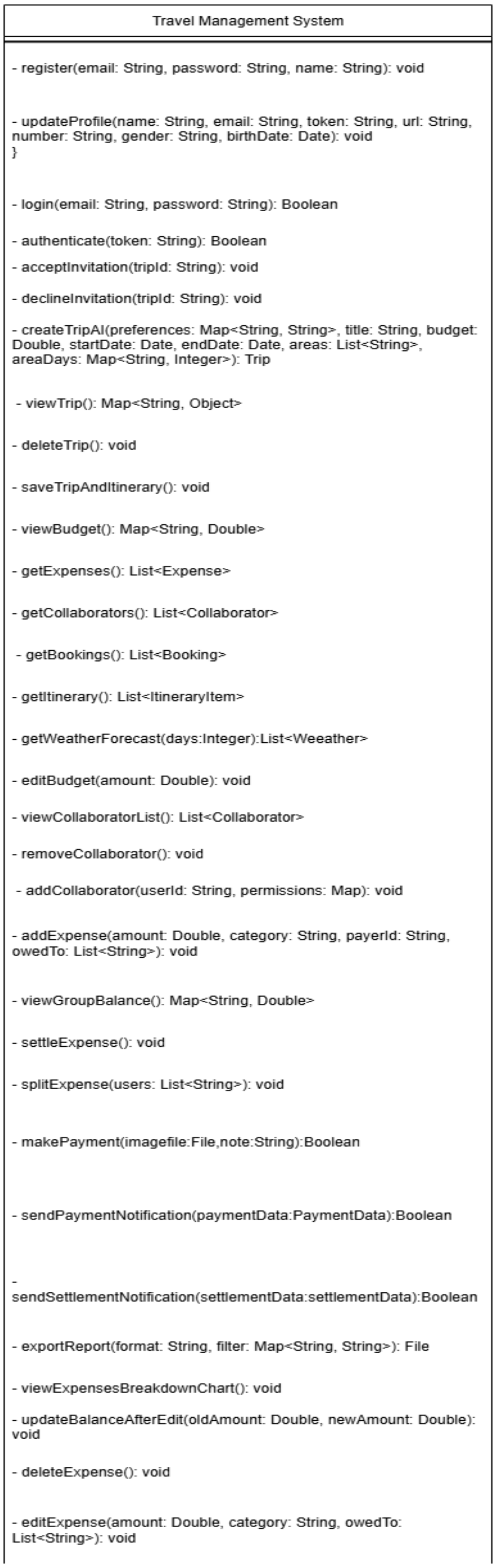
Figure 3.3.4 ERD

The Entity Relationship Diagram (ERD) models the data structure of the travel management system, capturing all key entities and their relationships. The central entity is Trips, which stores details such as title, duration, budget, number of travelers, and collaborators. Each trip is linked to Itinerary, which records the daily sequence of places visited, their categories, and weather integration. The system supports multiple place types, including Restaurants, Shops, Attractions, and Accommodations, all sharing common attributes such as placeId, name, address, category, rating, cost, and media references. These entities connect with the itinerary and bookings, where Bookings manage hotel reservations, confirmation codes, guest details, and pricing. Weather is associated with trips to provide forecast data for each day, while Expenses track financial details including category, amount, currency, and cost-splitting. The Users entity stores traveler profiles and preferences, with links to trips, payments, and area selections managed through AreaDays. The system also integrates Notifications, enabling updates and action prompts between senders and receivers within a trip context. Relationships are established through foreign keys to ensure data integrity, such as trips owning itineraries, bookings, weather records, and expenses, while places are referenced across itineraries and bookings. Overall, the ERD supports core functionalities including trip planning, place matching, budgeting, booking management, weather-aware itineraries, and collaborative travel planning.

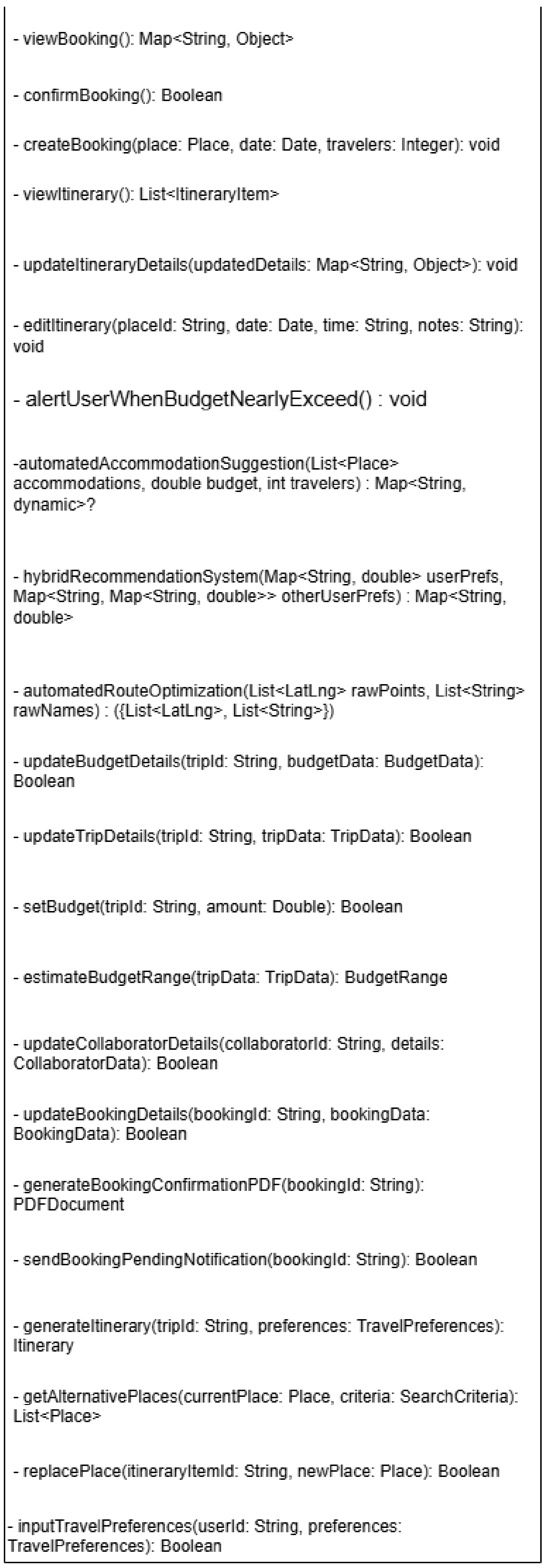
3.3.5 Class Diagram




- trip:Trip
-amount:Double
-payer:User
-settledDate:DateTime
-recipeint:User
-settledBy:User
-status:String
-bookingId: String
- trip:Trip
-accommodation:Accommodation
-checkInDate: Date
-checkOutDate: Date
- roomType: String
-status: String
-totalPrice: Double
-guests: Integer
-confirmationCode:String
-area: String
-accommodationId: String
-name: String
-roomTypes: Map<String, RoomInfo>
-availableRoomsByDate:Map<Date,RoomAvailability>
-checkInTime:String
-checkOutTime:String
-rating: Double
-price_level : Integer
-estimatedCost : Double
-location:LocationInfo
-address: String
-areaId: String
-itemId:String
-trip:Trip
-place:Place
-isOutdoor:Boolean
-scheduledDate:Date
-notificationId:String
-type:String
-sender:User
-recipient:User
-message:String
-status:String
-createdDate:DateTime
-relatedTrip :Trip
-actionData:Map<String,Object>
-weatherId:String
-condition:String
-date:Date



```
Travel Management System

- register(email: String, password: String, name: String): void

- updateProfile(name: String, email: String, token: String, url: String,
number: String, gender: String, birthDate: Date): void
}

- login(email: String, password: String): Boolean

- authenticate(token: String): Boolean

- acceptInvitation(tripId: String): void

- declineInvitation(tripId: String): void

- createTripAI(preferences: Map<String, String>, title: String, budget:
Double, startDate: Date, endDate: Date, areas: List<String>,
areaDays: Map<String, Integer>): Trip

 - viewTrip(): Map<String, Object>

- deleteTrip(): void

- saveTripAndItinerary(): void

- viewBudget(): Map<String, Double>

- getExpenses(): List<Expense>

- getCollaborators(): List<Collaborator>

 - getBookings(): List<Booking>

- getItinerary(): List<ItineraryItem>

- getWeatherForecast(days:Integer):List<Weeather>

- editBudget(amount: Double): void

- viewCollaboratorList(): List<Collaborator>

- removeCollaborator(): void

 - addCollaborator(userId: String, permissions: Map): void

- addExpense(amount: Double, category: String, payerId: String,
owedTo: List<String>): void

- viewGroupBalance(): Map<String, Double>

- settleExpense(): void

- splitExpense(users: List<String>): void

- makePayment(imagefile:File,note:String):Boolean

- sendPaymentNotification(paymentData:PaymentData):Boolean

-
sendSettlementNotification(settlementData:settlementData):Boolean

- exportReport(format: String, filter: Map<String, String>): File

- viewExpensesBreakdownChart(): void

- updateBalanceAfterEdit(oldAmount: Double, newAmount: Double):
void

- deleteExpense(): void

- editExpense(amount: Double, category: String, owedTo:
List<String>): void
```



- viewBooking(): Map<String, Object>

- confirmBooking(): Boolean

- createBooking(place: Place, date: Date, travelers: Integer): void

- viewItinerary(): List<ItineraryItem>

- updateItineraryDetails(updatedDetails: Map<String, Object>): void

- editItinerary(placeId: String, date: Date, time: String, notes: String): void

- alertUserWhenBudgetNearlyExceed() : void

-automatedAccommodationSuggestion(List<Place> accommodations, double budget, int travelers) : Map<String, dynamic>?

- hybridRecommendationSystem(Map<String, double> userPrefs, Map<String, Map<String, double>> otherUserPrefs) : Map<String, double>

- automatedRouteOptimization(List<LatLng> rawPoints, List<String> rawNames) : ({List<LatLng>, List<String>})

- updateBudgetDetails(tripId: String, budgetData: BudgetData): Boolean

- updateTripDetails(tripId: String, tripData: TripData): Boolean

- setBudget(tripId: String, amount: Double): Boolean

- estimateBudgetRange(tripData: TripData): BudgetRange

- updateCollaboratorDetails(collaboratorId: String, details: CollaboratorData): Boolean

- updateBookingDetails(bookingId: String, bookingData: BookingData): Boolean

- generateBookingConfirmationPDF(bookingId: String): PDFDocument

- sendBookingPendingNotification(bookingId: String): Boolean

- generateItinerary(tripId: String, preferences: TravelPreferences): Itinerary

- getAlternativePlaces(currentPlace: Place, criteria: SearchCriteria): List<Place>

- replacePlace(itineraryItemId: String, newPlace: Place): Boolean

- inputTravelPreferences(userId: String, preferences: TravelPreferences): Boolean

Figure 3.3.5 Class Diagram

The class diagram illustrates the fundamental structure and interactions of entities within the travel management system, following an object-oriented design approach that organizes related attributes and methods into cohesive classes. Each class represents a core entity in the system and encapsulates business logic relevant to its functional responsibilities. This architectural model serves as the foundation for data organization, API method implementations, and Firebase database integration throughout the application.

The User class manages personal information and authentication with essential attributes including userId, email, userName, displayName, phoneNumber, gender, dateOfBirth, and user-defined preferences. The class provides comprehensive authentication functionality through methods such as registerEmail(), loginEmail(), and updateProfile() for account management. Social collaboration features are implemented through acceptInvitation() and related invitation handling methods. Trip interaction capabilities include createTrip(), updateTripDetails(), and getTrips(), enabling users to manage their travel planning activities while maintaining proper ownership and collaboration relationships.

The Trip class serves as the primary orchestrator for travel planning operations, containing critical attributes such as tripId, title, startDate, endDate, budget, numberOfTravelers, areaDays, and collaboratorIds. This class implements the core automation logic identified throughout the system analysis. Key intelligent methods include automatedAccommodationSuggestion() which analyzes budget constraints and group size to recommend suitable lodging options, and hybridRecommendationSystem() that combines content-based and collaborative filtering algorithms to suggest optimal travel areas. The automatedRouteOptimization() method implements the 2-opt TSP algorithm for minimizing travel distances, while weatherForecastAppliedInItineraryPlanning() integrates meteorological data into trip recommendations. The alertUserWhenBudgetNearlyExceed() method provides proactive financial monitoring by notifying users when expenses approach or exceed budget thresholds.

The Place class models travel destinations with comprehensive attributes including placeId, name, address, area, category, description, coordinates, estimatedCost, and rating information. Functional methods such as searchPlace(), viewPlaceDetails(), and savePlaceToTrip() enable location discovery and trip integration. The ItineraryItem class represents individual schedule entries, linking places to specific trip contexts **through** itemId, tripId, placeId, date, time, and sequence attributes. The isMismatch boolean tracks weather-related scheduling conflicts, while methods like addItinerary(), updateItineraryDetails(), and viewItinerary() manage daily scheduling operations and support the complex itinerary generation workflows.

The Expense class implements sophisticated financial tracking with attributes including expenseId, amount, category, currency, payerId, isOthers, isSplit, splitType, and settlement status indicators. Core financial operations are managed through addExpense(), editExpense(), and splitExpenses() methods that support the complex expense sharing algorithms analyzed in previous use cases. The monitorTripDebt() method calculates and tracks group financial obligations, implementing the settlement optimization logic identified in the group balance management workflows. Additional financial methods include makePayment() for transaction processing and settleUp() for debt resolution between group members.

3.4 Implementation Challenges and Issues

1. WeatherAPI only provides 3-day forecast in free package

The free version of WeatherAPI restricts weather forecasting to just 3 days, which limits the system's ability to generate weather-aware trip plans for longer durations. This constraint reduced the accuracy of suggestions for multi-day itineraries.

2. Cannot use booking platform API to get real-time hotel prices

Major booking platforms restrict API access to enterprise partners or require substantial licensing fees, preventing real-time price integration for smaller applications and creating a gap between user expectations for current pricing and technical feasibility.

3. Google Places APIs not suggesting the most popular places

Google Places API did not consistently return the most well-known or highly rated attractions in the top results. This affected the quality of recommendations, requiring additional filtering and manual verification to improve relevance.

3.5 Project Timeline

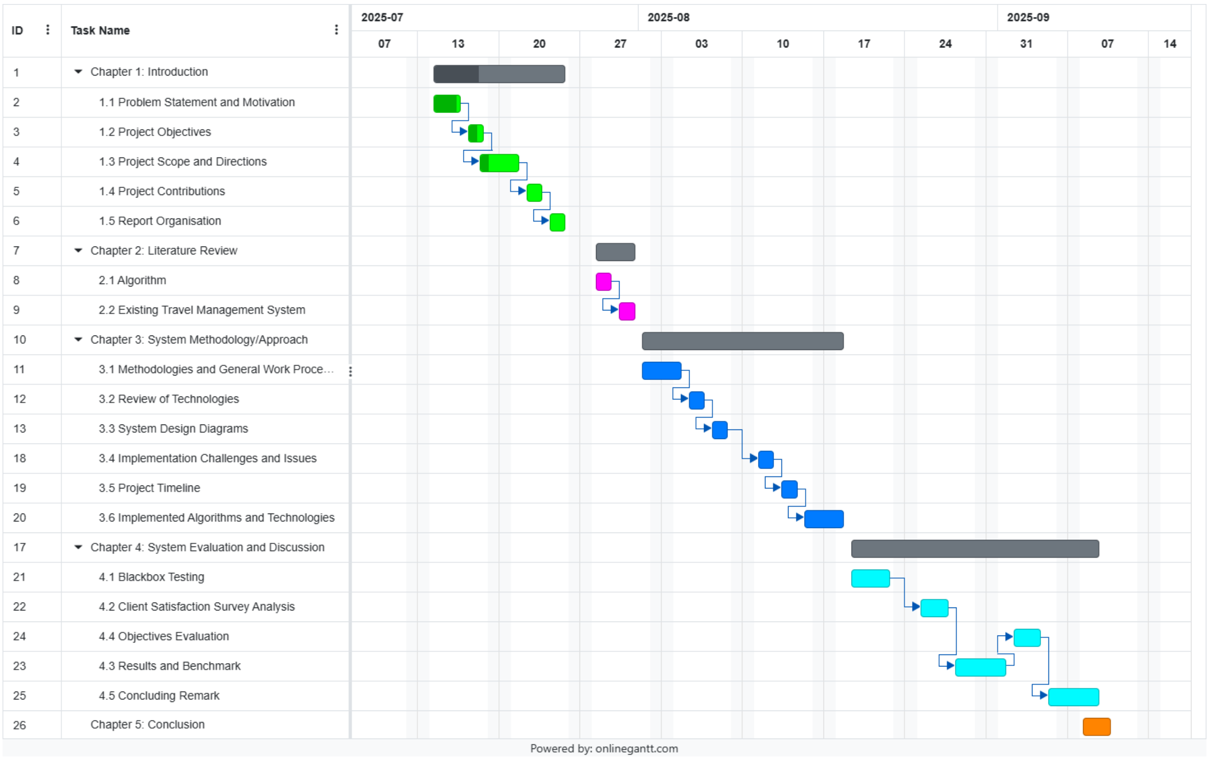


Figure 3.5 Gantt Chart

This project timeline spans from 14 July 2025 to 10 September 2025 and is structured into five main chapters, each broken down into specific tasks with defined durations and dependencies. Work begins with Chapter 1: Introduction (14–25 July), which is allocated 10 working days. Within this period, Problem Statement and Motivation is addressed first (14–16 July), followed by Project Objectives (17–18 July), Project Scope and Directions (18–21 July), and Project Contributions (22–23 July), before concluding with Report Organisation (24–25 July). Each section flows logically into the next, ensuring that the foundation of the report is firmly established. Afterward, Chapter 2: Literature Review is scheduled from 28–31 July, focusing on the Algorithm (28–29 July) and Existing Travel Management System (30–31 July), providing a strong theoretical background for the system.

The bulk of the work falls under Chapter 3: System Methodology/Approach, planned for 1–18 August. This technical chapter covers Methodologies and General Work Procedures (1–4 August), Review of Technologies (5–6 August), System Design Diagrams (7–8 August), Implementation Challenges and Issues (11–12 August), Project Timeline (13–14 August), and Implemented Algorithms and Technologies (15–18 August). Given its complexity, this chapter is given the longest duration to document development processes in detail. Following this, Chapter 4: System Evaluation and Discussion extends from 19 August to 9 September. It begins with Blackbox Testing (19–22 August), continues with Client Satisfaction Survey Analysis (25–27 August), then Results and Benchmark (28 August–1 September), Objectives Evaluation (2–4 September), and finally the Concluding Remark (5–9 September). These tasks are arranged sequentially to ensure that system evaluation is systematically carried out and thoroughly discussed.

Finally, Chapter 5: Conclusion is scheduled for 8–10 September, overlapping slightly with the final remark of Chapter 4, to summarize the overall findings and complete the report. Overall, the Gantt chart ensures a structured and time-bound workflow, moving from introductory groundwork and literature review to detailed methodology, system evaluation, and final conclusion. The timeline allocates more days to critical technical and evaluation sections while keeping introductory and concluding chapters concise, reflecting a balanced and realistic approach to completing the FYP report within the given timeframe.

3.6 Implemented Algorithms and Technologies

3.6.1 Hybrid Area Recommendation

This module ranks candidate areas by first estimating a Collaborative Filtering (CF) score from similar users' behaviors, then a Content-Based Filtering (CBF) score from the current user's preferences vs. area profiles, and finally combining them with adaptive weights.

(A) Collaborative Filtering

Each user u is represented by a preference vector $x_u = (p_u(f))_{f \in F}$,where $pu(f)$is the user's score for feature f (e.g., beach, city, history, food). The similarity between two users u and v is measured using **cosine similarity**:

$$\mathrm{sim}(u, v) = \frac{x_u \cdot x_v}{||x_u||\ ||x_v||}$$

Only neighbors with $sim(u, v) \geq 0.5$ are retained. For each area a, let $d_{v,a}$, be the number of **picked days** for neighbor v (0 if not picked). The CF score is computed as:

$$C_a = \sum_{\mathrm{sim}(u,v) \geq 0.5} \mathrm{sim}(u, v)\ d_{v,a}$$

$$W_a = \sum_{\mathrm{sim}(u,v) \geq 0.5} |\,\mathrm{sim}(u, v)|$$

The per-area CF score is then:

$$s_{CF}(u, a) = \begin{cases} \mathrm{clamp}_{[0,10]}\left(\frac{C_a}{W_a}\right), & W_a > 0 \\ \text{(no CF)}, & W_a = 0 \end{cases}$$

(B) Content-Based Filtering

Each area a has a profile vector $w_a(f)$, representing how strongly feature f applies to that area. Each user u has a preference score $p_u(f)$ for the same features (default 5.0 if unknown). The raw dot product is:

$$s_{CBF}^{\mathrm{raw}}(u, a) = \sum_{f \in F} p_u(f)\ w_a(f)$$

To ensure comparability, the raw score is normalized to a 0–10 scale using a maximum constant $M = 8 \cdot 8 \cdot 4$:

$$s_{CBF}(u,a) = \text{clamp}_{[0,10]}\left(\frac{s_{CBF}^{\text{raw}}(u,a)}{M} \cdot 10\right)$$

(C) Adaptive Fusion

The system adaptively combines CF and CBF depending on whether CF evidence exists:

$$S(u,a) = \begin{cases} \text{clamp}_{[0,10]}\big(0.7\, s_{CF}(u,a) + 0.3\, s_{CBF}(u,a)\big), & W_a > 0 \\ \text{clamp}_{[0,10]}\big(0.5\, s_{CBF}(u,a)\big), & W_a = 0 \end{cases}$$

If similar users exist (Wa>0W_a > 0Wa>0), the system gives **70% weight to CF** and **30% to CBF**.

If no similar users exist, it penalizes unseen areas by using only **half of the CBF score**

3.6.2 Budget Range Estimation

This module predicts a user-specific total trip budget by analyzing historical trips, normalizing them to a per-person-per-night (PPN) cost, and aggregating them with similarity weights. The output is a budget estimate accompanied by a confidence band that reflects the uncertainty of the match.

(A) Context similarity & compatibility filters

The system first filters historical trips to retain only those that are reasonably comparable to the current user's planned trip. Compatibility is determined through:

- **Area similarity** $Sarea(u,r) \in [0,1]$; measured by day-weighted overlap of selected areas.
- **Hotel-level weight** $w_{hotel}(u,r) \in [0,1]$; only trips with comparable hotel level are kept; discard if <0.2.
- **Nights** weight $w_{nights}(u,r) \in [0,1]$; only trips with a comparable duration; discard if <0.3.
- **Room-type compatibility** (must be true to keep the trip).

Additional soft weights:

- **Traveller-count weight** $w_{trav}(u,r)$
- **Day-count weight** $w_{day}(u,r)$

The overall weight for trip r is the **product**:

$$w_r = S_{\text{area}}(u,r)\, w_{\text{trav}}(u,r)\, w_{\text{day}}(u,r)\, w_{\text{hotel}}(u,r)\, w_{\text{nights}}(u,r)$$

(B) Normalize to per-person-per-night (PPN)

To make costs comparable, each trip is normalized into a **per-person-per-night** value:

$$\mathrm{PPN_r} = \frac{\mathrm{budget_r}}{\max(1,\mathrm{trav_r})\ \max(1,\mathrm{nights_r})}$$

(C) Robust outlier trimming

The system removes outliers by trimming to the 10th and 90th percentiles $(\mathrm{P_{10}}, \mathrm{P_{90}})$ of all PPN values:

$$\mathrm{P_{10}} \leq \mathrm{PPN_r} \leq \mathrm{P_{90}}$$

(D) Weighted averaging

If some trips survive and $\sum_r w_r > 0$, the weighted PPN is:

$$\widehat{\mathrm{PPN}} = \frac{\sum_r w_r\ \mathrm{PPN_r}}{\sum_r w_r}$$

Otherwise, fall back to the **unweighted mean**:

$$\widehat{\mathrm{PPN}} = \frac{1}{|R|}\sum_{r\in R} \mathrm{PPN_r}$$

(E) Scale to the user's total & post-adjustments

Let T be **myTravellers**, N be **myNights**.

$$\mathrm{TotalEstimate} = \widehat{\mathrm{PPN}}\cdot T \cdot N$$

Group-size economy. Apply a multiplicative adjustment

$$\mathrm{TotalEstimate} \leftarrow g(T)\ \cdot \mathrm{TotalEstimate}$$

where $g(T) \leq 1$captures shared-cost effects.

(F) Range construction (confidence band)

Depending on match strictness, set a **band**:

$$\mathrm{BudgetRange} = \begin{cases} 0.05, & \text{if exact match on (travellers, days, hotel level)} \\ 0.15, & \text{otherwise} \end{cases}$$

and form the range:

$$BudgetRange = [(1-\beta)TotalEstimate, (1+\beta)TotalEstimate]$$

3.6.3 Route Optimization Algorithm

The day plan starts at the accommodation (fixed start) and visits each selected place once. Distances are computed using the Haversine great-circle formula.

(A) Haversine distance

For points $i = (\phi_i, \lambda_i)$ and $j = (\phi_j, \lambda_j)$ in radians:

$$d_{ij} = 2r\ arcsin\left(\sqrt{\sin^2\left(\frac{\Delta\emptyset}{2}\right) + \cos(\phi_i)\cos(\phi_j)\sin^2\left(\frac{\Delta\lambda}{2}\right)}\right)$$

With $\Delta\phi = \phi_j - \phi_i, \Delta\lambda = \lambda_j - \lambda_i, r \approx 6371$ km

The system precomputes a full distance matrix $D = [d_{ij}]$

(B)Objective

Given a visiting order $\pi = (\pi 1, \dots, \pi n)$with $\pi_1 = h$(the hotel), the objective is to minimize:

$$\text{Dist}(\pi) = \sum_{t=1}^{n-1} d_{\pi_t,\pi_{t+1}}$$

(C) Construction and improvement

The algorithm generates an initial route with the hotel fixed at the start. It then applies **2-opt local search**: for edges $(i, i+1)$ and $(j, j+1)$perform a swap if

$$d_{i,i+1} + d_{j,j+1} > d_{i,j} + d_{i+1,j+1}$$

and accept improvements until no further reduction in distance is possible.

(D)Result

The output is an optimized sequence of points and names, the minimized travel distance, and the number of improvements performed. If returnToStart is enabled, the tour is closed by appending the hotel at the end.

3.6.4 Automated Accommodation Suggestions

This module filters hotels by **level (rating range)**, finds a **feasible room arrangement** given travellers and capacities, **optimizes by cost-per-person**, validates against the **trip budget**, then picks a hotel and builds **alternatives within 2 km** at the same price level.

(A) Level filter (rating range)

Each hotel with rating r_h is accepted only if it falls within the selected level's range

$$r_{min} \leq r_h \leq r_{max}$$

(B) Price floor by level (quality guardrails)

Each room type must meet the minimum price requirement associated with the hotel level L

$$p_{\text{single}} \geq p_{\text{single}}^{min}(L), p_{\text{family}} \geq p_{\text{family}}^{min}(L)$$

(C) Room Arrangement Feasibility

- **Small groups (T ≤ 2):** Prefer single room if feasible; otherwise fallback to family.
- **Medium groups (3 ≤ T ≤ 4):** Require family room.
- **Large groups (T ≥ 5):** Calculate multiple room options.

$$T \leq c_{\text{single}}, \quad T \leq c_{\text{family}}$$

(D) Large Group Options

Family-only option:

$$k_f = \left\lceil \frac{T}{c_{\text{family}}} \right\rceil, \quad \text{Cost}_{\text{f}} = k_f \, p_{\text{family}}$$

Single-only option:

$$k_s = \left\lceil \frac{T}{c_{\text{single}}} \right\rceil, \quad \text{Cost}_{\text{s}} = k_s \, p_{\text{single}}$$

Mixed option (family + single):

$$k_f = \left\lfloor \frac{T}{c_{\text{family}}} \right\rfloor, T_{\text{rem}} = T - k_f \, c_{\text{family}}, k_s = \left[\frac{T_{\text{rem}}}{c_{\text{single}}} \right]$$

$$\text{Cost}_{\text{mixed}} = k_f \, p_{\text{family}} + k_s \, p_{\text{single}}$$

E) Cost per Person and Optimal Selection

The optimal arrangement minimizes the **cost per person (CPP):**

$$\text{CPP} = \frac{\text{Cost}}{T}, \quad \arg min \ \text{CPP}$$

(F) Total Accommodation Cost and Budget Validation

The per-night cost is scaled by the number of nights N:

$$\text{AccTotal} = \text{PricePerNight(total)} \times N$$

A hotel is accepted only if it satisfies:

$$\text{AccTotal} \leq \text{Budget}$$

(G) Alternate Hotel Selection

Finally, alternate hotels are retrieved if they share the same price level and are within **2 km** of the chosen hotel:

$$d(h_{\text{alt}}, h_{\text{sel}}) \leq 2 \text{ km}$$

3.6.5 Weather Forecast

he system returns a daily weather condition string for a given $(lat, lon, date)$. It uses a two-stage strategy:

1. Short range (≤ 3 days): call WeatherAPI and select the noon condition.
2. Extended range (≥ 4 days): generate conditions from seasonal/monsoon patterns (historical logic) and show a disclaimer.

(A) Day offset and branch selection

Rendered:

$$\text{daysAhead} = (\text{date} - \text{today})_{\text{days}} + 1$$
$$\text{if daysAhead} < 1 \Rightarrow \text{“Date in the past.”}$$
$$\text{if daysAhead} \leq 3 \Rightarrow \text{use API}; \quad \text{else} \ \Rightarrow \text{use seasonal model}$$

(B) Short-range forecast (API, daysAhead ≤ 3)

From WeatherAPI we get hourly conditions for that day. We pick the **hour closest to 12:00** (noon).

Rendered:

$$h^* = \arg\min_{h \in \{0,\ldots,23\}} |h - 12| \ \Rightarrow \text{Condition} = \text{cond(lat,lon,date,} h^*)$$

Rain reminder trigger

If the chosen condition indicates precipitation, show a reminder.

Rendered:

$$\text{if Condition} \in \{\text{Rain, Showers, Thunderstorm, …}\} \Rightarrow \text{ShowRainReminder}(\quad)$$

(C) Extended forecast (seasonal model, daysAhead ≥ 4)

When beyond the reliable API window, the system uses month/day seasonal patterns and monsoon type to generate a plausible daily condition.

Rendered:

$$m = \text{month(date)}, \quad d = \text{day(date)}$$

$$\text{Condition} \sim f_{\text{seasonal}}(m, d;\ \text{enhancedLogic} = 1)$$

$$\big(\text{monsoon}(m),\ \text{season}(m)\big) \leftarrow \text{patterns}[m]$$

Extended-range disclaimer

Show once per trip when using the seasonal branch.

Rendered:

$$\text{if first use in trip} \Rightarrow \text{ShowDisclaimer(“out of 3-day range”)}$$

(D) Overall piecewise definition

Rendered:

$$\text{Weather(lat,lon,date)} = \begin{cases} "\{“Date\ in\ the\ past.”\}, & \{daysAhead\} < 1 \\ \{cond\}(lat, lon, date, h^*), & 1 \le \{daysAhead\} \le 3 \\ f_{seasonal}(m, d), & \{daysAhead\} \ge 4 \end{cases}$$

Chapter 4
System Evaluation and Discussion

4.1 Blackbox Testing

4.1.1 Planning Module

4.1.1.1 Preference Dialog Tests

Test Scenario	**Input Data**	**Expected Output**	**Actual Output**	**Pass/Fail**	**Notes**
Valid preference selection	Select "City" and "Beach"	Preferences saved dialog closes	Preferences saved dialog closes	**Fail**	Dialog requires two clicks to close - Flutter framework issue.
No preferences selected	Click "confirm" without selecting any	Warning: "Please select at least 1 preference before continuing!"	Warning: "Please select at least 1 preference before continuing!"	**Pass**	
Single preference selected	Select only "Nature"	Preferences saved dialog closes	Preferences saved dialog closes	**Fail**	Dialog requires two clicks to close - Flutter framework issue.
All preferences selected	Select all 4 categories	Preferences saved dialog closes	Preferences saved dialog closes	**Fail**	Dialog requires two clicks to close - Flutter framework issue.

Table 4.1.1.1 Preference Dialog Tests

4.1.1.2 Trip Title Input Test

Test Scenario	Input Data	Expected Output	Actual Output	Pass/Fail	Notes
Valid Trip title	"Family holiday in Penang"	Title accepted	Title accepted	**Pass**	
Empty title	""	Generate itinerary button disabled	Generate itinerary button disabled	**Pass**	

Table 4.1.1.2 Trip Title Input Test

4.1.1.3 Area Selection Test

Test Scenario	Input Data	Expected Output	Actual Output	Pass/Fail	Notes
Valid selection	Select "George town" with 2 days	Area added to selectedAreas	Area added to selectedAreas	**Pass**	
Multiple Area selection	Select "Georgetown" 2 days and "Batu Ferringhi" 1 day	Area added to selectedAreas	Area added to selectedAreas	**Pass**	
Minimum days selection	Select "Georgetown"1 day	Area added to selectedAreas	Area added to selectedAreas	**Pass**	
Empty areas selection	Not select any areas	Generate itinerary button disabled	Generate itinerary button disabled	**Pass**	

Table 4.1.1.3 Area Selection Test

4.1.1.4 Area Days Validation

Test Scenario	**Input Data**	**Expected Output**	**Actual Output**	**Pass/Fail**	**Notes**
Slider minimum boundary	Move slider to minimum position	Slider shows 1 day,value=1	Slider shows 1 day,value=1	**Pass**	
Slider maximum boundary	Move slider to maximum position	Slider shows 5 day,value=5	Slider shows 5 day,value=5	**Pass**	
Slider interaction	Drag slider to middle position	Slider responds smoothly and shows correct day count	Slider responds smoothly and shows correct day count	**Pass**	
Slider default value	Open days selection dialog	Slider default to 1 day	Slider default to 1 day	**Pass**	

Table 4.1.1.4 Area Days Validation

4.1.1.5 Date Range Tests

Test Scenario	**Input Data**	**Expected Output**	**Actual Output**	**Pass/Fail**	**Notes**
Past date selection	Select yesterday to tomorrow	Should not be possible (firstDate = now)	**Past dates is disable**	**Pass**	
Date range mismatch	Select 3 areadays but select 2 days	Error "Date range mismatch"	Error "Date range mismatch"	**Pass**	
Date range match	Select 3 areadays and select 3 days	Date range accepted	Date range accepted	**Pass**	
Single day trip	Start date = end date	1 day trip accepted	**Date range accepted**	**Pass**	
Far future dates	Select dates in 2026	Date range accepted	Date range accepted	**Pass**	
Empty date range	Do not select any dates	Generate itinerary button disabled	Generate itinerary button disabled	**Pass**	

Table 4.1.1.5 Date Range Tests

4.1.1.6 Budget Input Test

Test Scenario	**Input Data**	**Expected Output**	**Actual Output**	**Pass/Fail**	**Notes**
Valid budget amount	Budget = 1000	Budget accepted	Budget accepted	Pass	
Zero budget	Budget = 0	Generate itinerary button disabled	Generate itinerary button disabled	**Pass**	
Negative budget	Budget = -500	Error:"Invalid budget"	Error:"Invalid budget"	**Pass**	
Decimal budget	Budget = 1000.10	Budget accepted	Budget accepted	**Pass**	
Large budget amount	Budget = 99999	Budget accepted	Budget accepted	**Pass**	
Non-numeric budget	Buget = "abc"	Error:"Only numeric character is accepted"	Error:"Only numeric character is accepted"	**Pass**	
Minimum budget	Budget <80	Error :"Minimum budget amount is RM 80"	Error :"Minimum budget amount is RM 80"	**Pass**	

Table 4.1.1.6 Budget Input Test

4.1.1.7 Traveler Count Test

Test Scenario	**Input Data**	**Expected Output**	**Actual Output**	**Pass/Fail**	**Notes**
Valid traveler count	Travelers = 2	Count accepted	Count accepted	**Pass**	
Large group	Travelers = 7	Count accepted	Count accepted	**Pass**	
Zero traveler	Travelers = 0	Generate itinerary button disabled	Generate itinerary button disabled	**Pass**	
Negative traveler	Travelers = -1	Generate itinerary button disabled	Generate itinerary button disabled	**Pass**	

Table 4.1.1.7 Traveler Count Test

4.1.1.8 Hotel Level Selection Test

Test Scenario	**Input Data**	**Expected Output**	**Actual Output**	**Pass/Fail**	**Notes**
Level 1 hotel selection	selectedHotelLevel =1	Hotels with rating below 3.0	Hotels with rating below 3.0	**Pass**	
Level 2 hotel selection	selectedHotelLevel =2	Hotels with rating 3.0-3.4	Hotels with rating 3.0-3.4	**Pass**	
Level 3 hotel selection	selectedHotelLevel =3	Hotels with rating 3.5-3.9	Hotels with rating 3.5-3.9	**Pass**	
Level 4 hotel selection	selectedHotelLevel =4	Hotels with rating 4.0-4.4	Hotels with rating 4.0-4.4	**Pass**	
Level 5 hotel selection	selectedHotelLevel =5	Hotels with rating 4.5-5.0	Hotels with rating 4.5-5.0	**Pass**	

Table 4.1.1.8 Hotel Level Selection Test

4.1.1.9 Room Type Matching Test

Test Scenario	**Input Data**	**Expected Output**	**Actual Output**	**Pass/Fail**	**Notes**
Single room for 1-2 travelers	Travelers=2, budget adequate	Single room selected	Single room selected	**Pass**	
Family room for >2 travelers	Travelers=3, budget adequate	Family room selected	Family room selected	**Pass**	
Capacity validation	Travelers =6,family room capacity =4	One single room and one family room selected	Error :"No hotels found matching the selected level or budget"	**Fail**	Room capacity logic error - system should reject hotels when family room capacity (4) < travelers (6), but shows generic 'no hotels found' message instead of specific capacity error.
Price validation	Roomprice > budget cap	Should not have this problem because minimum budget is minimum roomprice(RM80)	Do not have this problem	**Pass**	

Table 4.1.1.9 Room Type Matching Test

4.1.1.10 Trip Generation Prerequisites

Test Scenario	**Input Data**	**Expected Output**	**Actual Output**	**Pass/Fail**	**Notes**
Complete Valid inputs	All fields filled correctly	Trip generation proceeds	Trip generation proceeds	**Pass**	
Missing trip title	Title is null	Generate Itinerary button is disabled	Generate Itinerary button is disabled	**Pass**	
Missing areas	No areas selected	Generate Itinerary button is disabled	Generate Itinerary button is disabled	**Pass**	
Missing dates	startDate or endDate null	Generate Itinerary button is disabled	Generate Itinerary button is disabled	**Pass**	
Missing budget	Budget = null	Error :"Minimum budget amount is RM 80"	Error :"Minimum budget amount is RM 80"	**Pass**	

Table 4.1.1.10 Trip Generation Prerequisites

4.1.1.11 Plan Style Configuration Test

Test Scenario	**Input Data**	**Expected Output**	**Actual Output**	**Pass/Fail**	**Notes**
Food-heavy plan	SelectedPlanStyle = foodHeavy	1 attractions,4 meals,1 shop per day	1 attractions,4 meals,1 shop per day	**Pass**	
Balanced plan	SelectedPlanStyle = balanced	2 attractions, 2meals,2 shops per day	2 attractions, 2meals,2 shops per day	**Pass**	
Attraction-seeking plan	SelectedPlanStyle =attractionSeeking	4 attractions, 2meals,0 shops per day	4 attractions, 2meals,0 shops per day	**Pass**	
Shopping-lover plan	SelectedPlanStyle = shoppingLover	1 attractions, 2meals,3 shops per day	1 attractions, 2meals,3 shops per day	**Pass**	

Table 4.1.1.11 Plan Style Configuration Test

4.1.1.12 Weather API Tests

Test Scenario	Input Data	Expected Output	Actual Output	Pass/Fail	Notes
Current weather(1-3days)	Date within next 3 days	WeatherAPI data retrieved	WeatherAPI data retrieved	**Pass**	
Extended forecast(>3 days)	Date beyond 3 days	Historical pattern weather generated	Historical pattern weather generated	**Pass**	
Weather API failure	API returns error	Return unknown weather	Return unknown weather	**Pass**	
Invalid coordinates	Lat/lon out of range	API error handled gracefully	API error handled gracefully	**Pass**	

Table 4.1.1.12 Weather API Tests

4.1.1.13 Weather -based Filtering Test

Test Scenario	Input Data	Expected Output	Actual Output	Pass/Fail	Notes
Rainy weather filtering	Weather = "Heavy Rain"	Indoor places prioritized	Indoor places prioritized	**Pass**	
Sunny weather filtering	Weather = "Sunny"	All places(indoor+outdoor) included	All places(indoor+outdoor) included	**Pass**	
Unknown weather handling	Weather = "unknown"	All attractions included	All attractions included	**Pass**	
Rain reminder display	Weather contains rain	Umbrella reminder snackbar shown	Umbrella reminder snackbar shown	**Pass**	

Table 4.1.1.13 Weather -based Filtering Test

4.1.1.14 Save Trip Test

Test Scenario	**Input Data**	**Expected Output**	**Actual Output**	**Pass/Fail**	**Notes**
Complete trip saved	Save All trip data present	Message: Trip saved to Firebase successfully	Message: Trip saved to Firebase successfully	**Pass**	
Direct Booking	User clicks 'save trip' button	A bottom sheet will show up with the options of 'book in app' or 'skip for now'	A bottom sheet will show up with the options of 'book in app' or 'skip for Passnow'	**Pass**	
Missing trip data	Some required fields null	Error message displayed	Error message displayed	**Pass**	

Table 4.1.1.14 Save Trip Test

4.1.1.15 Map Interaction Tests

Test Scenario	**Input Data**	**Expected Output**	**Actual Output**	**Pass/Fail**	**Notes**
Hotel pin tap	Tap red hotel marker	Hotel details sheet displayed	Hotel details sheet displayed	**Pass**	
Attraction pin tap	Tap numbered attraction	Place details sheet displayed	Place details sheet displayed	**Pass**	
Alternative hotel tap	Tap black alternative marker	Alternative hotel sheet displayed	Alternative hotel sheet displayed	**Pass**	
Map toggle functionality	Toggle "Show Alternate Hotels"	Markers appear/disappear correctly	Markers appear/disappear correctly	**Pass**	
Map refresh after chanegs	Update hotel selection	Map markers update correctly	Map markers update correctly	**Pass**	

Table 4.1.1.15 Map Interaction Tests

4.1.1.16 Replace with alternative places test

Test Scenario	Input Data	Expected Output	Actual Output	Pass/Fail	Notes
No clashing with places in itinerary	User click “use instead” button	Show dialog 'Replaced stop #$index on Day ${day + 1} with "$newName".'	Show dialog 'Replaced stop #$index on Day ${day + 1} with "$newName".'	**Pass**	
clashing with places in itinerary	User click “use instead” button	Show dialog “This place has already included in your current itinerary.Do you want to replace it?”	Show dialog “This place has already included in your current itinerary.Do you want to replace it?”	**Pass**	
Insist to replace if it is clashing with current itinerary	User clicks “Use anyway”	Replace the places with the clashed places	Replace the places with the clashed places	**Pass**	
Do not replace if it is clashing with current itinerary	User clicks “No”	Show “Kept your original stop”	Show “Kept your original stop”	**Pass**	

Table 4.1.1.16 Replace with alternative places test

4.1.1.17 TSP Route Optimization Test

Test Scenario	Input Data	Expected Output	Actual Output	Pass/Fail	Notes
Small route (2 points)	Hotel + 1 stop	Route returned as-is, no optimization	Route `[0,1]` returned, distance=1.0	**Pass**	
Medium route (5 points)	Hotel + 4 stops	Route optimized using 2-opt	Optimized route ≤ initial distance	**Pass**	
Large route (10 points)	Hotel + 9 stops	Route optimized with improvements logged	Optimized route found, **improvement count not ≥1 (test failed)**	**Fail**	Algorithm optimized distance but did not log improvements consistently (possible logic/tolerance issue).
0 points	No places	Empty route returned safely	Empty route `[]` returned	**Pass**	
Fixed start optimization	fixedStart = 0	Hotel remains at position 0	First element = 0	**Pass**	

Table 4.1.1.17 TSP Route Optimization Test

4.1.1.18 Distance calculations

Test Scenario	Input Data	Expected Output	Actual Output	Pass/Fail	Notes
Same location distance	point1 = point2	Distance = 0.0 km	Distance = 0.0 km	**Pass**	
Known distance calculation	George Town → Batu Ferringhi	Distance ≈ 11–13 km	Distance ≈ 12 km	**Pass**	
Large distance calculation	Penang (George Town) → Kuala Lumpur	Distance ≈ 350+ km (road approximation)	Distance ≈ 352 km	**Pass**	
Invalid coordinates	Invalid lat/lng values	Error handled gracefully	ArgumentError thrown & caught gracefully	**Pass**	

Table 4.1.1.18 Distance calculations

4.1.1.19 Place Matching Tests

Test Scenario	**Input Data**	**Expected Output**	**Actual Output**	**Pass/Fail**	**Notes**
Exact name match	"Penang Hill" in database	Exact match found and used	"Penang Hill"	**Pass**	
Fuzzy name match	"Penang Hill" vs "Bukit Penang"	Fuzzy match ≥ 60% → "Penang Hill"	No match at 0.6	**Fail**	Word-order + synonym ("Bukit"≠"Hill") caused rating < 0.6; needs threshold adjustment or synonym handling
No match found	Completely different name	No match (added to unmatched_places)	Null (unmatched)	**Pass**	
Multiple similar names	Several 60%+ matches in database	First/best match selected	One of the similar names	**Pass**	

Table 4.1.1.19 Place Matching Tests

4.1.1.20 Alternative Suggestions Tests

Test Scenario	Input Data	Expected Output	Actual Output	Pass/Fail	Notes
Attraction alternatives	Request alternatives for museum	Similar attractions within 2 km	3 nearby museums returned; sorted by distance	**Pass**	
Restaurant alternatives	Request alternatives for restaurant	Similar restaurants within radius	"Local Seafood"	**Pass**	
Shop alternatives	Request alternatives for souvenir shop	Similar shops within radius	"Souvenir Central"	**Pass**	
No alternatives available	Remote location with no nearby places	Empty alternatives list	Empty list	**Pass**	
Duplicate prevention	Alternative overlaps with selected	Duplicate excluded from results	"Pinang Peranakan Mansion" excluded	**Pass**	

Table 4.1.1.20 Alternative Suggestions Tests

4.1.1.21 Budget Range Estimation Test

Test Scenario	Input Data	Expected Output	Actual Output	Pass/Fail	Notes
Sufficient similar trips	5+ similar trips in DB	Weighted average budget calculated	Weighted avg between lowest & highest ppd	**Pass**	
Exact match preference	Same travelers & days available	Exact matches used with 10% band	2000 ±10% → 1800–2200	**Pass**	
No similar trips	No trips in DB	Fallback budget calculation used	Returned 200	**Pass**	
Outlier filtering	Some extreme budget values	Outliers excluded from calculation	5000 excluded; 1500 & 2000 retained	**Pass**	
User similarity calc	Compare preference vectors	Cosine similarity calculated	≈0.85 (high similarity)	**Pass**	

Table 4.1.1.21 Budget Range Estimation Test

4.1.1.22 Trip Display & Navigation

Test Case	**Input Data**	**Expected Output**	**Actual Output**	**Pass/Fail**	**Notes**
Load My Trips Screen	Valid user ID	Display list of user's trips (owned + collaborated)	Display list of user's trips (owned + collaborated)	**Pass**	
Empty Trips Display	User with no trips	Show "No trips yet. Let's create one"message with explore icon	Show "No trips yet. Let's create one"message with explore icon	**Pass**	
Trip Card Display	Trip with complete data	Show trip card with title, dates, budget,	Show trip card with title, dates, budget,	**Pass**	
Trip Owner Indicator	User owns trip	Display delete button on trip card	Display delete button on trip card	**Pass**	
Collaborator View	User is collaborator	show trip card without delete options	show trip card without delete options	**Pass**	
Trip Tap Navigation	Tap on trip card	Navigate to TripDetailScreen with correct tripId	Navigate to TripDetailScreen with correct tripId	**Pass**	
Invalid Trip Data	Trip with missing/corrupted data	Show "Error loading trip" or "Invalid trip data" message	Show "Error loading trip" or "Invalid trip data" message	**Pass**	

Table 4.1.1.22 Trip Display & Navigation

4.1.1.23 Trip Deletion Test

Test Case	Input Data	Expected Output	Actual Output	Pass/Fail	Notes
Delete Confirmation Dialog	Tap delete button	Show confirmation dialog	Show confirmation dialog	**Pass**	
Cancel Deletion	Tap “Cancel” in dialog	Dialog closes, trip remains	Dialog closes, trip remains	**Pass**	
Confirm Deletion	Tap “Delete” in dialog	Trip deleted, success snackbar shown	Trip deleted, success snackbar shown	**Pass**	
Delete Error Handling	Deletion fails	Show “Error deleting trip” snackbar	Show “Error deleting trip” snackbar	**Pass**	
Delete Non-Owner	Collaborator tries to delete	Delete button not visible/accessible	Delete button not visible/accessible	**Pass**	

Table 4.1.1.23 Trip Deletion Test

4.1.1.24 Itinerary Management

Test Case	Input Data	Expected Output	Actual Output	Pass/Fail	Notes
Load Itinerary Tab	Valid trip ID	Display itinerary items grouped by date	Display itinerary items grouped by date	**Pass**	
Day Selection	Tap on different day	Switch to selected day's itinerary	Switch to selected day's itinerary	**Pass**	
Place Card Display	Itinerary item	Show place with image, name, category, weather info	Show place with image, name, category, weather info	**Pass**	
Place Details	Tap on place card	Open Place Details Sheet with place information	Open Place Details Sheet with place information	**Pass**	
Empty Itinerary	Day with no places	Show "No places planned for this day"	Show "No places planned for this day"	**Pass**	
Weather Integration	Valid itinerary item	Display weather condition for the day	Display weather condition for the day	**Pass**	
Invalid Place Data	Corrupted place data	Show "Place Not Found" with error icon	Show "Place Not Found" with error icon	**Pass**	

Table 4.1.1.24 Itinerary Management

4.1.2 Budgeting Module

4.1.2.1 Expense Dialog Test

Test Case	Input Data	Expected Output	Actual Output	Pass/Fail	Notes
Add Expense Dialog	Tap add expense button	Show expense dialog with all fields	Show expense dialog with all fields	**Pass**	
Valid Expense Entry	Amount: 100, Category: Food	Expense saved successfully	Expense saved successfully	**Pass**	
Invalid Amount	Amount: -50 or empty	Show "Please enter a valid amount" error	Show "Please enter a valid amount" error	**Pass**	
Split Expense Toggle	Toggle split option	Show/hide split settings based on trip type	Show/hide split settings based on trip type	**Pass**	
Custom Split Validation	Custom split not equal to total	Show "Custom split amounts must equal total amount"	Show "Custom split amounts must equal total amount"	**Pass**	
Solo Trip Split	Solo trip with split enabled	Split option disabled/hidden	Split option disabled/hidden	**Pass**	
Edit Existing Expense	Tap edit on expense	Populate dialog with existing data	Populate dialog with existing data	**Pass**	
Delete Expense	Tap delete and confirm	Expense removed, success message shown	Expense removed, success message shown	**Pass**	
Budget Warning	Expenses reach 80% of budget	Send notification to alert budget exceed	Send notification to alert budget exceed	**Pass**	
Budget Adjustment	Adjust budget in warning dialog	Budget updated successfully	Budget updated successfully	**Pass**	
No expenses breakdown chart	No expenses added yet	Show no expenses to show yet	Show no expenses to show yet	**Pass**	

Table 4.1.2.1 Expense Dialog Test

4.1.2.2 Settlement system

Test Case	**Input Data**	**Expected Output**	**Actual Output**	**Pass/Fail**	**Notes**
Settlement Dashboard	Group trip with expenses	Display individual balances and suggested settlements	Display individual balances and suggested settlements	**Pass**	
Payment Receipt Upload	User click pay button and Upload valid image	Receipt uploaded, pending status shown	Receipt uploaded, pending status shown	**Pass**	
Invalid Receipt Upload	Upload fails	Show error message for upload failure	Show error message for upload failure	**Pass**	
Settle Payment	Tap "Settle" button	Show settlement confirmation dialog	Show settlement confirmation dialog	**Pass**	
Mark as Settled	Confirm settlement	Create settlement record, update status	Create settlement record, update status	**Pass**	
Pending Status Display	Receipt uploaded but not settled	Show "Pending" status instead of "Pay" button	Show "Pending" status instead of "Pay" button	**Pass**	
Solo Trip Settlement	Solo trip	Settlement dashboard not displayed	Settlement dashboard not displayed	**Pass**	
View Receipt	Tap "View" on receipt	Open receipt viewer dialog with image	Open receipt viewer dialog with image	**Pass**	

Table 4.1.2.2 Settlement system

4.1.3 Collaborator Module

Test Case	**Input Data**	**Expected Output**	**Actual Output**	**Pass/Fail**	**Notes**
Add Collaborator Dialog	Tap add collaborator button	Show search dialog with username field	Show search dialog with username field	**Pass**	
Search Users	Enter valid username	Display matching users in search results	Display matching users in search results	**Pass**	
No Search Results	Enter non-existent username	Show "No users found" message	Show "No users found" message	**Pass**	
Send Collaboration Request	Tap send icon	Send notification, show success message	Send notification, show success message	**Pass**	
Search Timeout	Network timeout during search	Show "Error searching users" message	Show "Error searching users" message	**Pass**	
Remove Collaborator	Owner removes collaborator	Show confirmation, remove from trip	Show confirmation, remove from trip	**Pass**	
Non-Owner Remove Attempt	Collaborator tries to remove others	Remove button not visible	Remove button not visible	**Pass**	
Display Owner	Trip owner in list	Show "Owner" label, no remove button	Show "Owner" label, no remove button	**Pass**	

Table 4.1.3 Collaborator Module

4.1.4 Booking Module

Test Case	**Input Data**	**Expected Output**	**Actual Output**	**Pass/Fail**	**Notes**
Load Booking Tab	Trip with booking	Display booking card with hotel details	Display booking card with hotel details	**Pass**	
No Booking State	Trip without accommodation	Show "No accommodation yet" empty state	Show "No accommodation yet" empty state	**Pass**	
Pending Booking Action	Booking status: PENDING	Show "Complete booking" button	Show "Complete booking" button	**Pass**	
Confirmed Booking Action	Booking status: CONFIRMED	Show "Download Confirmation" button	Show "Download Confirmation" button	**Pass**	
Generate PDF	Tap download confirmation	Generate and display booking PDF	Generate and display booking PDF	**Pass**	
Booking	Tap "Details"	Open booking details	Open booking details	**Pass**	

Table 4.1.4 Booking Module

4.1.5 Notification Module

Test Case	**Input Data**	**Expected Output**	**Actual Output**	**Pass/Fail**	**Notes**
Load notifications screen	User with notifications	Display notifications in sections	Display notifications in sections	**Pass**	
Load empty notifications	User with no notifications	Show "No notifications yet" message	Show "No notifications yet" message	**Pass**	
Accept collaborator invite	Tap "Accept" on invite	Success message, user added to trip	Success message, user added to trip	**Pass**	
Decline collaborator invite	Tap "Decline" on invite	Decline message, user not added	Decline message, user not added	**Pass**	
View payment receipt	Tap "View Receipt"	Receipt image dialog opens	Receipt image dialog opens	**Pass**	
Settle payment	Tap "Settle Up" on payment	Payment marked as settled	Payment marked as settled	**Pass**	
Adjust budget warning	Tap "Adjust Budget"	Budget edit dialog opens	Budget edit dialog opens	**Pass**	
Review booking notification	Tap "Review booking"	Booking page opens	Booking page opens	**Pass**	
Refresh notifications	Tap refresh button	"Booking status refreshed" message	"Booking status refreshed" message	**Pass**	
Handle network error	No internet connection	"Error loading notifications" message	"Error loading notifications" message	**Pass**	

Table 4.1.5 Notification Module

4.2 Client Satisfaction Survey Analysis

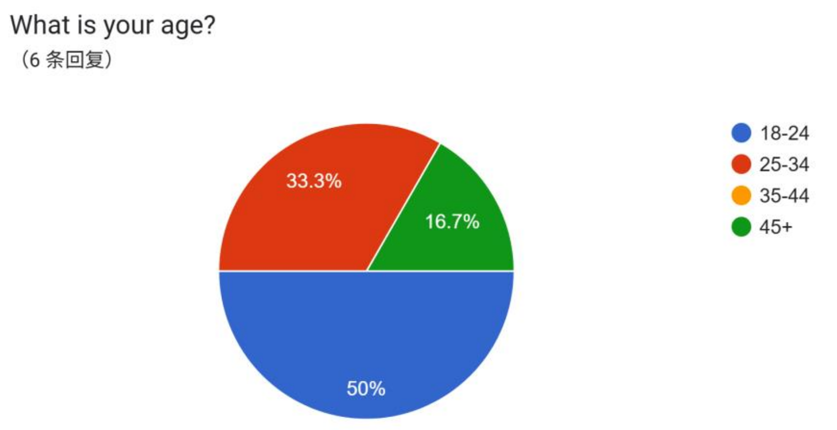


Figure 4.2.1 Age distribution of respondents

The age distribution of respondents shows that half of the participants (50%) fall within the 18–24 age group, followed by 33.3% in the 25–34 group, and 16.7% are aged 45 and above, while no respondents were in the 35–44 category. This indicates that the majority of ExploreEasy’s potential users are young adults, mainly university students and young professionals, who are more open to using digital solutions for travel planning. Since this group is often budget conscious and tech-savvy, the system should focus on affordability, convenience, and mobile-first design to meet their needs effectively.

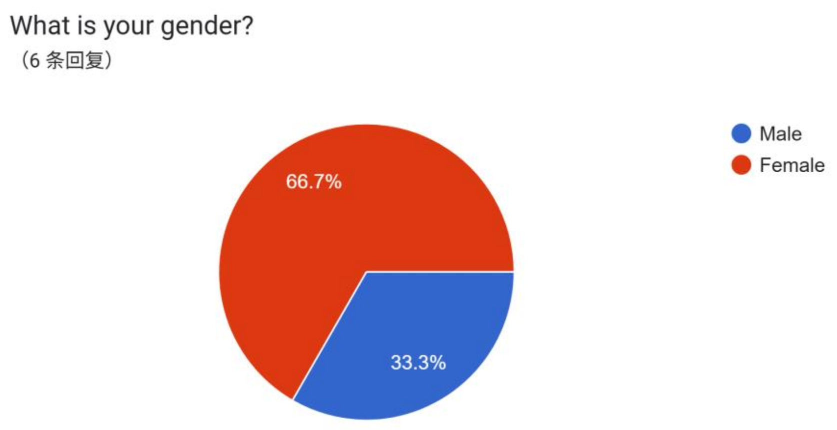


Figure 4.2.2 Gender distribution of respondents

The gender breakdown reveals that 66.7% of the respondents are female, compared to 33.3% male. This suggests that female travelers may form a larger portion of ExploreEasy’s target users. Therefore, the design and features of the application should take into account factors that are typically prioritized by female travelers, such as safety, budget transparency, and accommodation reliability, while still ensuring inclusivity for male users. This highlights the importance of creating a balanced interface that appeals to both genders.

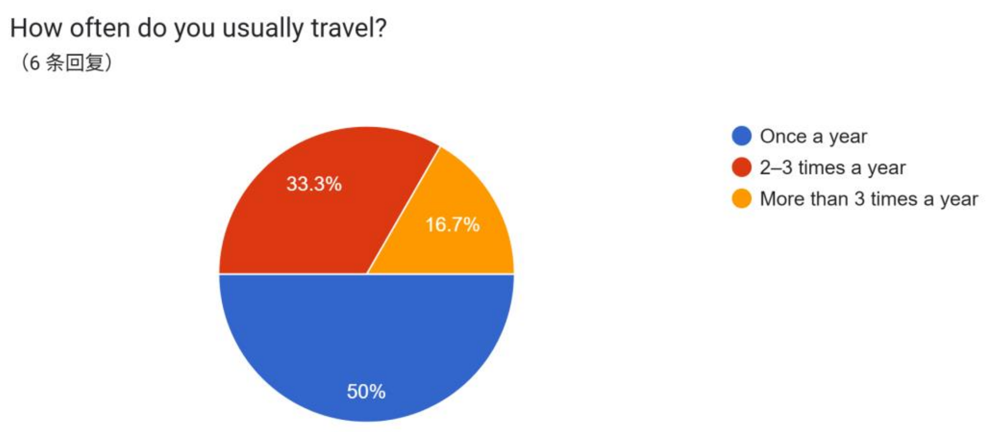


Figure 4.2.3 Frequency of respondents' travel

In terms of travel frequency, the results show that 50% of respondents travel once a year, 33.3% travel two to three times a year, while only 16.7% travel more than three times annually. This indicates that most users are occasional travelers rather than frequent ones. As a result, ExploreEasy should aim to provide features that simplify the travel planning process, such as automated itinerary generation, budget estimation, and weather-aware recommendations. These features will help occasional travelers save time and effort in planning their limited trips.

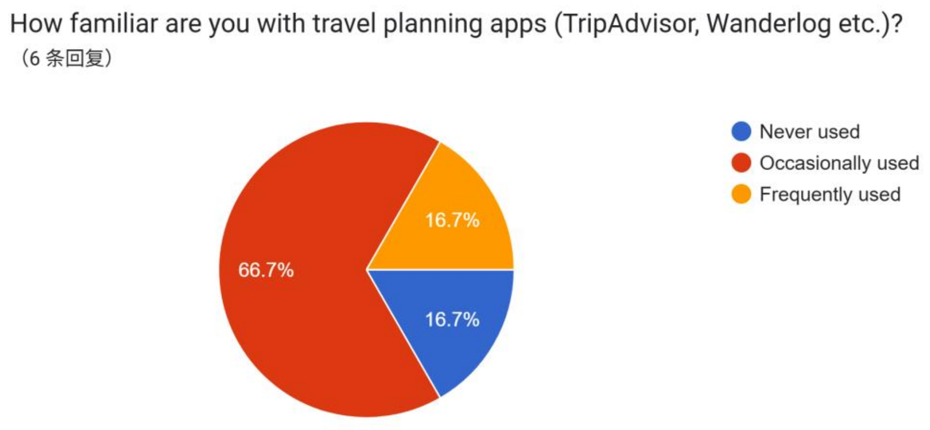


Figure 4.2.4 Familiarity of respondents with travel planning apps

The pie chart shows that majority of 66.7% reported occasional usage, 16.7% had never used such applications, and only 16.7% were frequent users. This demonstrates that most respondents have some awareness but are not highly reliant on existing tools. Therefore, ExploreEasy has the opportunity to position itself as a beginner-friendly yet innovative platform, combining simplicity of use with unique functions such as budget alerts, smart accommodation suggestions, and itinerary optimization based on weather conditions.

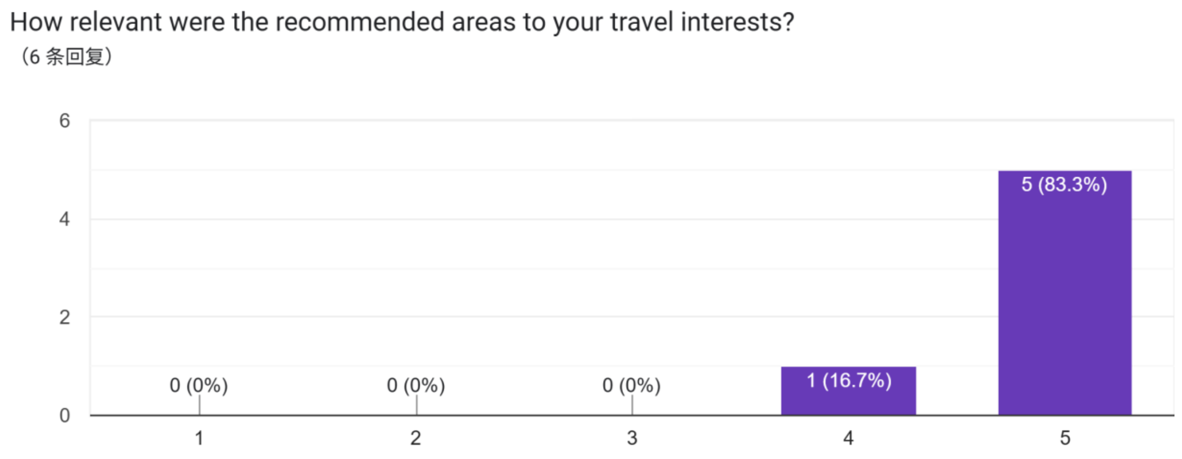


Figure 4.2.5 Relevance of recommended areas to respondents' travel interests

This chart shows that the majority of respondents (83.3%) rated the recommended areas as highly relevant to their travel interests, while the remaining 16.7% rated them slightly less relevant but still positive. None of the participants gave a low rating. This indicates that the area recommendation algorithm in ExploreEasy is effective in matching users' preferences with suitable destinations, showing strong alignment between the system's suggestions and user expectations.

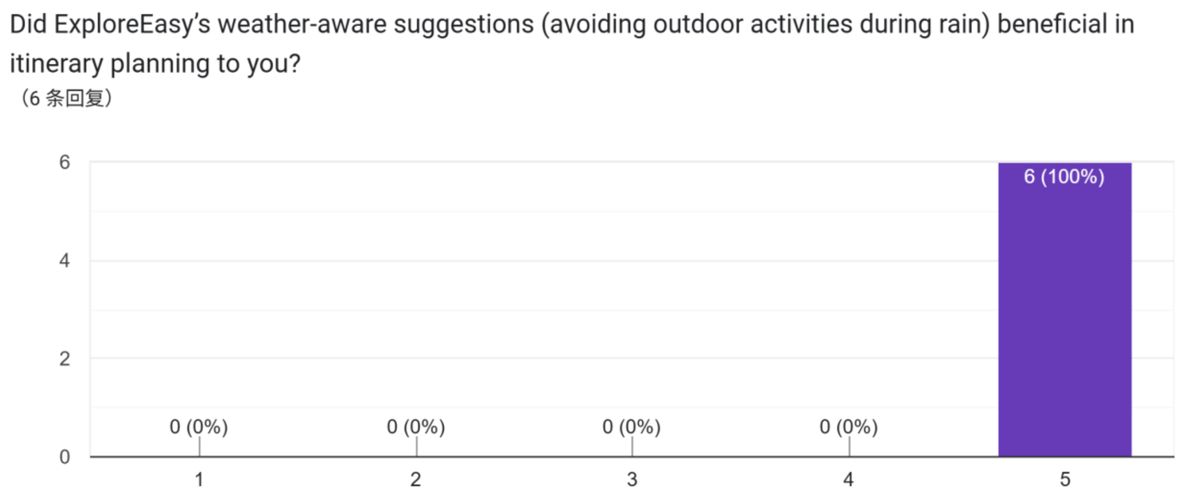


Figure 4.2.6 Respondents' feedback on weather-aware suggestions in itinerary planning

This chart shows that 100% of respondents agreed that the weather-aware suggestions, which avoid recommending outdoor activities during rain, were beneficial in itinerary planning. The unanimous positive response highlights the importance and usefulness of integrating real-time weather data into the planning process.

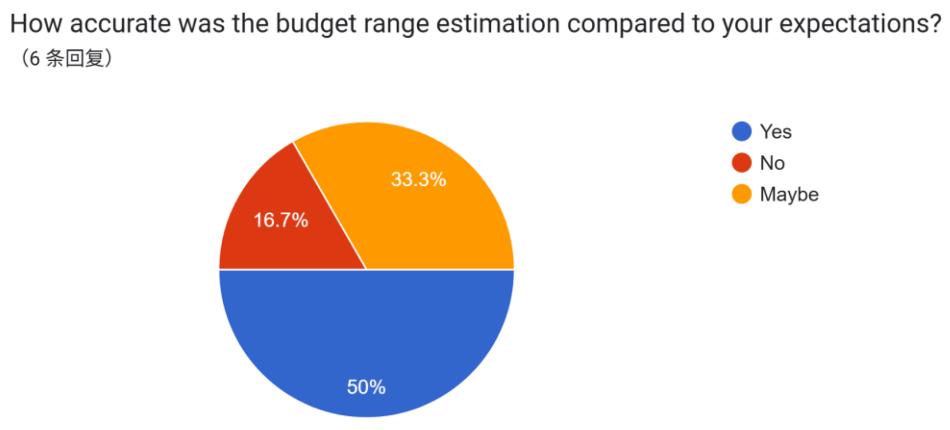


Figure 4.2.7 Accuracy of budget range estimation compared to respondents' expectations

This chart shows that half of the respondents (50%) agreed that the budget range estimation was accurate, while 33.3% selected "Maybe," and 16.7% responded "No." The majority therefore recognized the usefulness of this feature, but the presence of uncertainty and a small portion of dissatisfaction indicates that while the system performs well overall, there is still room for refinement in providing more precise or personalized estimations.

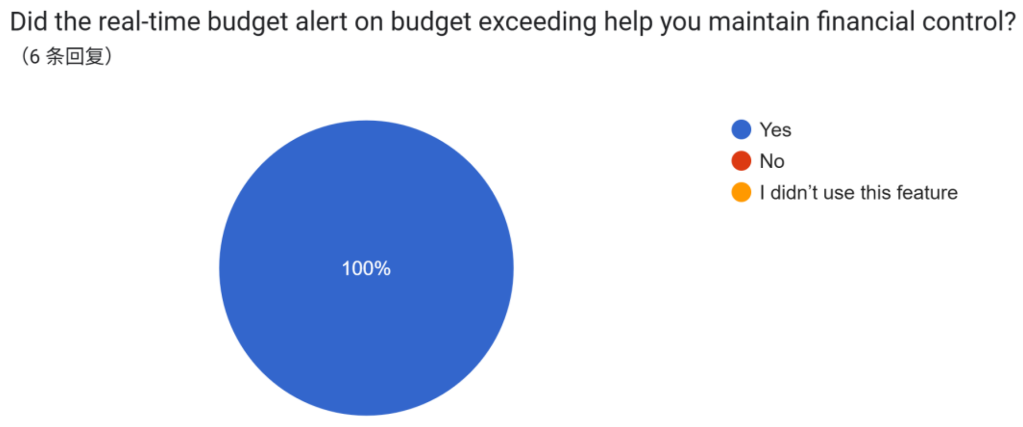


Figure 4.2.8 Respondents' feedback on real-time budget alerts for financial control

This chart shows that 100% of respondents agreed that the real-time budget alert feature helped them maintain financial control during their trip planning. The strong positive feedback highlights the effectiveness of this function in preventing overspending and supporting users in staying within their financial limits. This confirms that budget monitoring is a highly valuable feature for travelers.

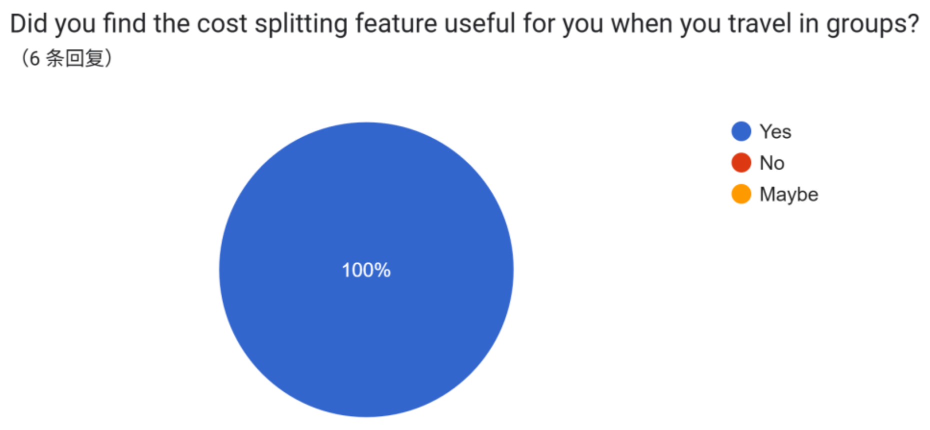


Figure 4.2.9 Usefulness of cost splitting feature for group travel

This chart shows that 100% of respondents found the cost splitting feature useful when traveling in groups. The unanimous positive response emphasizes the importance of this function in simplifying group travel expense management. By automatically dividing costs among participants, ExploreEasy reduces potential conflicts and makes financial arrangements more transparent and convenient.

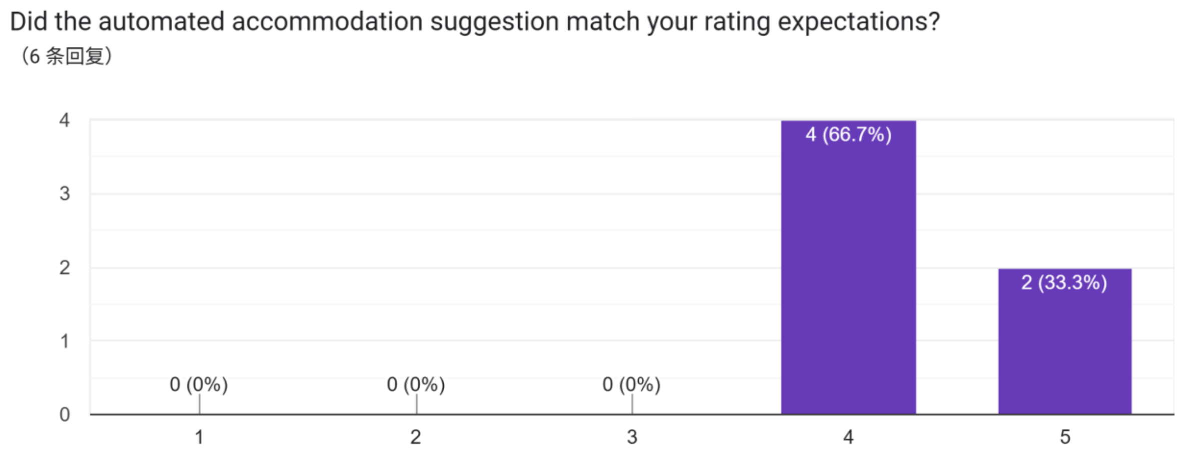


Figure 4.2.10 Accuracy of automated accommodation suggestions against respondents' expectations

This chart shows that 66.7% of respondents rated the automated accommodation suggestion at level 4, and 33.3% gave it the highest rating of 5. No negative ratings were recorded, which demonstrates that the majority of users were satisfied with the quality of automated hotel recommendations. This confirms the effectiveness of the feature, though continuous improvement can further enhance confidence and accuracy.

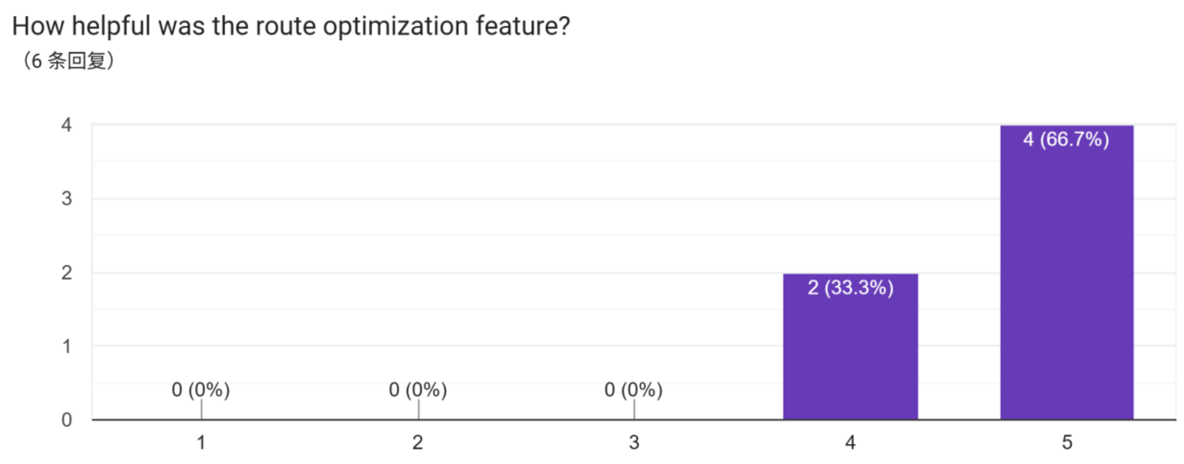


Figure 4.2.11 Respondents' feedback on the helpfulness of the route optimization feature

This chart shows that 66.7% of respondents rated the route optimization feature as very helpful, while 33.3% rated it as helpful. No respondents provided negative feedback, indicating strong approval of this feature. This confirms that ExploreEasy's route planning successfully minimizes travel distance and time, thereby improving convenience and efficiency in the travel experience.

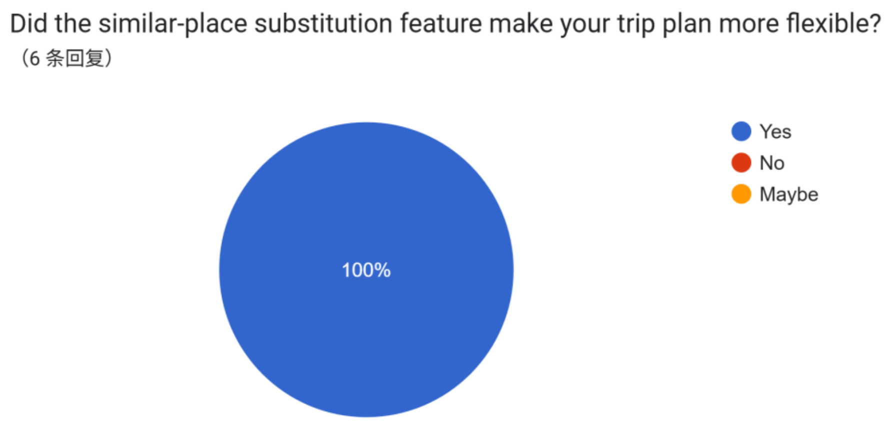


Figure 4.2.12 Respondents' feedback on the flexibility of similar-place substitution feature

This chart shows that 100% of respondents agreed that the similar-place substitution feature improved flexibility in their trip plan. The unanimous positive response highlights the importance of offering alternatives when certain attractions are unavailable or less suitable.

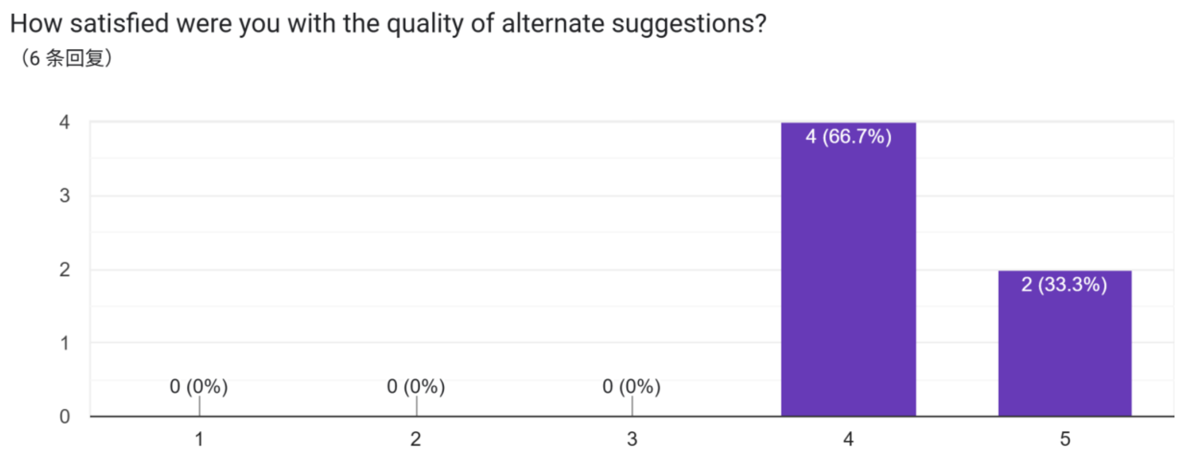


Figure 4.2.13 Satisfaction with the quality of alternate suggestions

This chart shows that 66.7% of respondents rated their satisfaction with the quality of alternate suggestions at level 4, while 33.3% rated it at the highest level of 5. This indicates that users are generally very satisfied with the system’s ability to provide meaningful alternative recommendations, ensuring flexibility and adaptability in their travel plans.

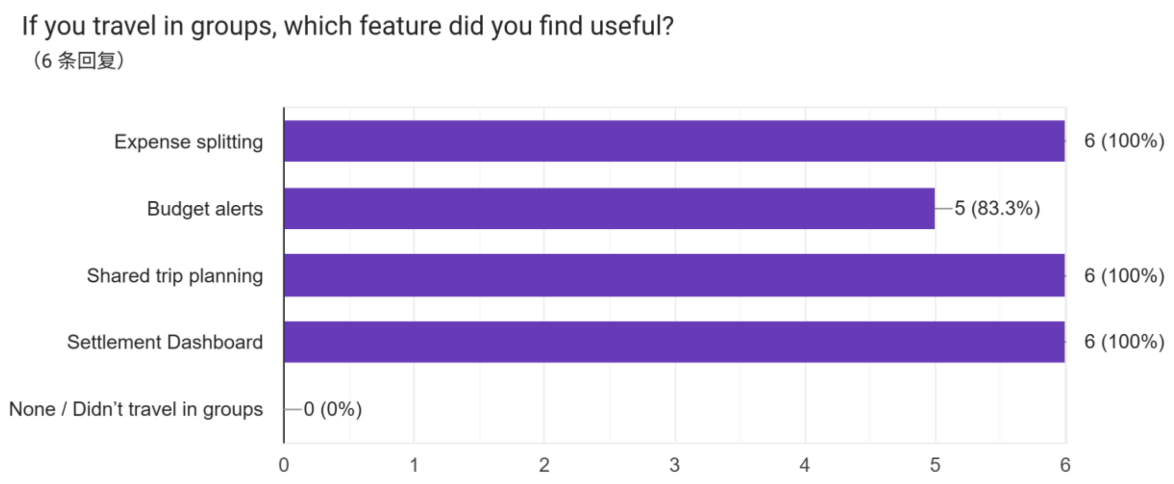


Figure 4.2.14 Usefulness of group travel features (expense splitting, budget alerts, shared trip planning, settlement dashboard)

This chart shows that all respondents (100%) found the group-related features such as expense splitting, shared trip planning, and the settlement dashboard useful, while 83.3% also highlighted the budget alerts as valuable. None of the participants reported that these features were unnecessary. This reflects that ExploreEasy’s group travel functions are highly effective in meeting user needs for financial transparency, collaboration, and convenience when traveling with others.

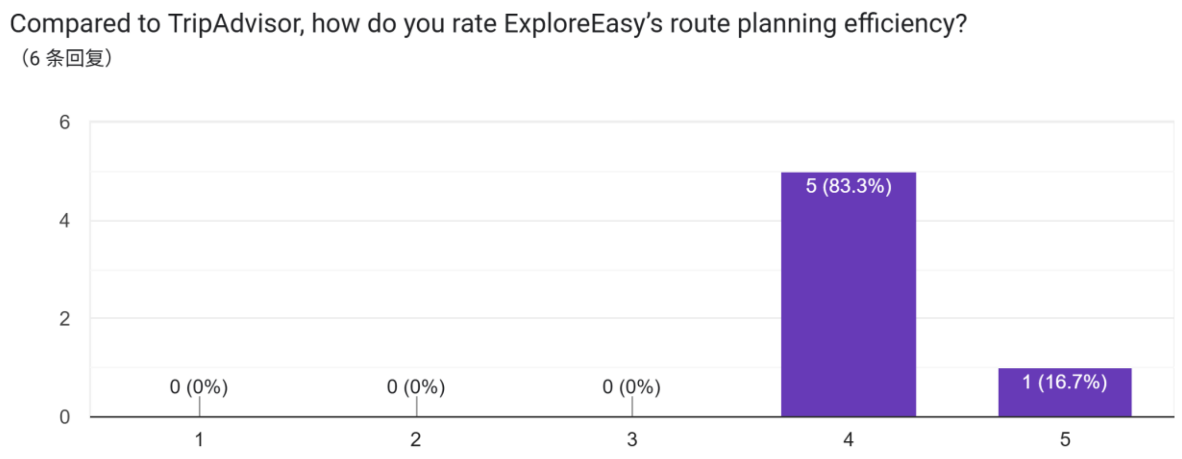


Figure 4.2.15 Comparison of ExploreEasy's route planning efficiency against TripAdvisor

This chart shows that 83.3% of respondents rated ExploreEasy's route planning efficiency as 4 out of 5, while 16.7% gave it a perfect score of 5. No negative ratings were received, which suggests that users perceive ExploreEasy as highly efficient, and in some cases superior, when compared to TripAdvisor's route planning.

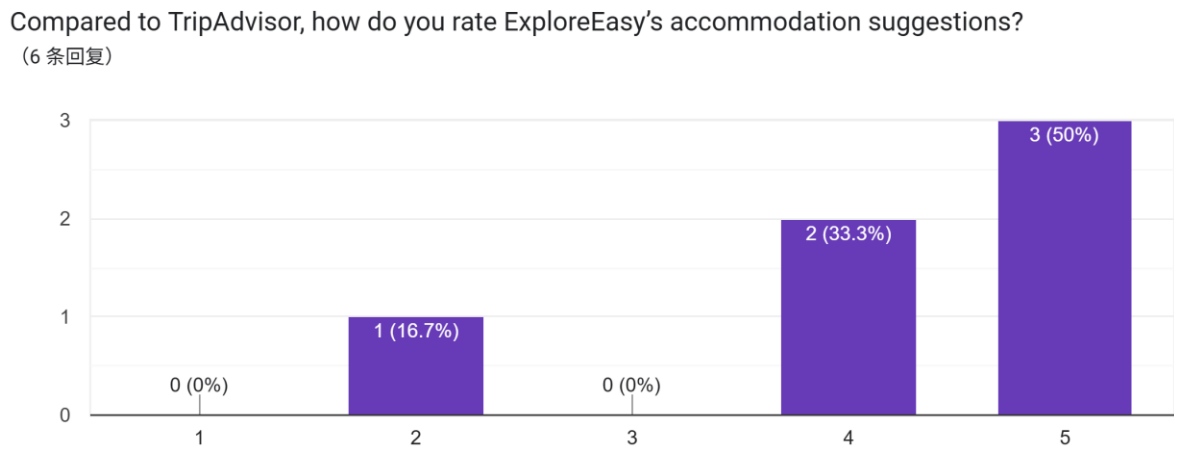


Figure 4.2.16 Comparison of ExploreEasy's accommodation suggestions against TripAdvisor

This chart shows that 50% of respondents rated ExploreEasy's accommodation suggestions at the highest level of 5, 33.3% rated them at level 4, while 16.7% gave a lower score of 2. While most users were satisfied with the feature, the presence of some dissatisfaction indicates that further improvements are needed to ensure consistency in meeting rating expectations. This chart shows that 50% of respondents rated ExploreEasy’s budget management features at level 5, 16.7% at level 4, while 33.3% rated it relatively low at level 2. This suggests that although the majority recognized the strength of the system’s budget functions, there is a subset of users who felt that Wanderlog may have better implementation. This highlights an area where ExploreEasy can refine its budget management tools to further enhance reliability and accuracy.

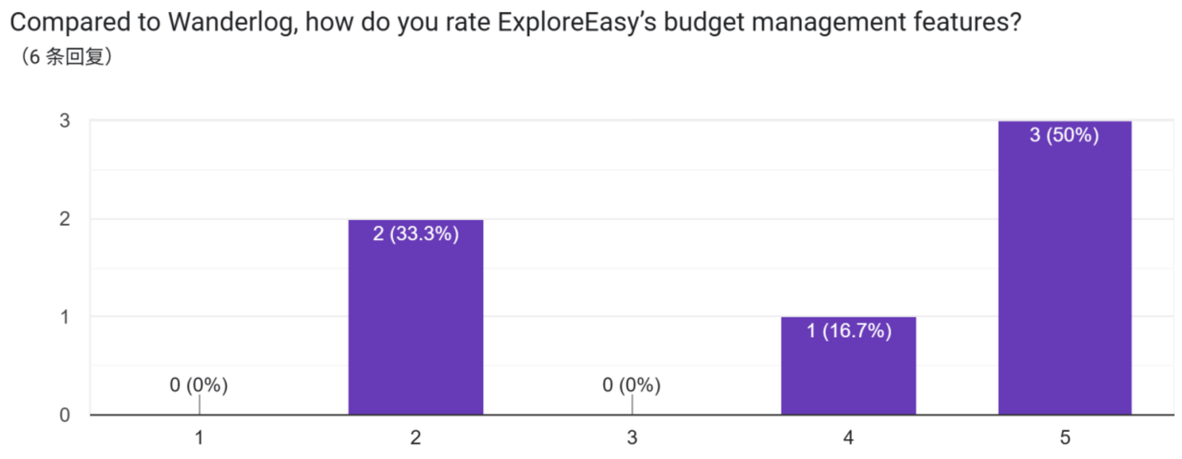


Figure 4.2.17 Comparison of ExploreEasy’s budget management features against Wanderlog

This chart shows that 50% of respondents rated ExploreEasy’s budget management features at level 5, 16.7% at level 4, while 33.3% rated it relatively low at level 2. This suggests that although the majority recognized the strength of the system’s budget functions, there is a subset of users who felt that Wanderlog may have better implementation. This highlights an area where ExploreEasy can refine its budget management tools to further enhance reliability and accuracy.

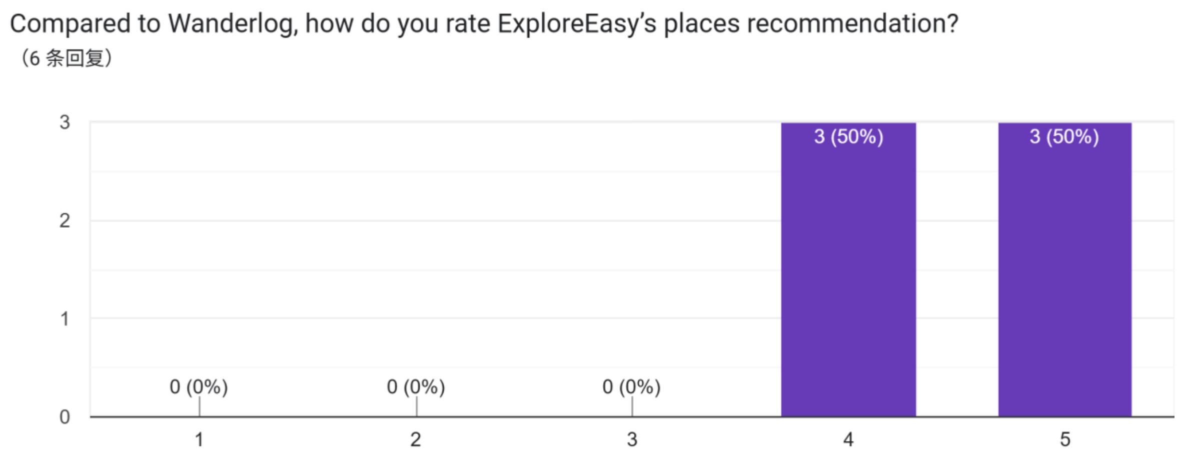


Figure 4.2.18 Comparison of ExploreEasy's place recommendations against Wanderlog

This chart shows that 50% of respondents rated ExploreEasy's place recommendation feature as very good (level 5), while the remaining 50% rated it at level 4. None of the respondents rated it poorly, which confirms that ExploreEasy's recommendation system is competitive with Wanderlog and consistently meets user expectations.

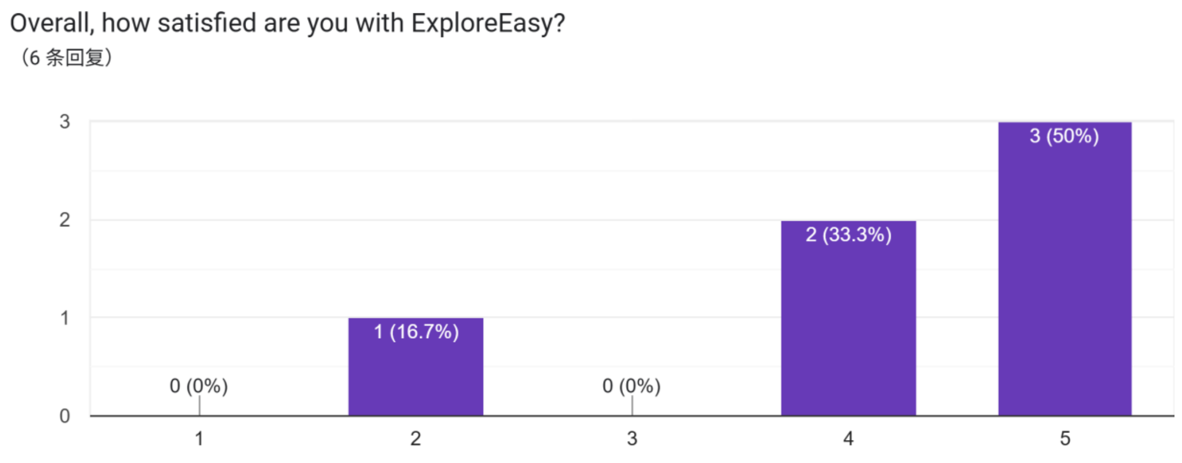
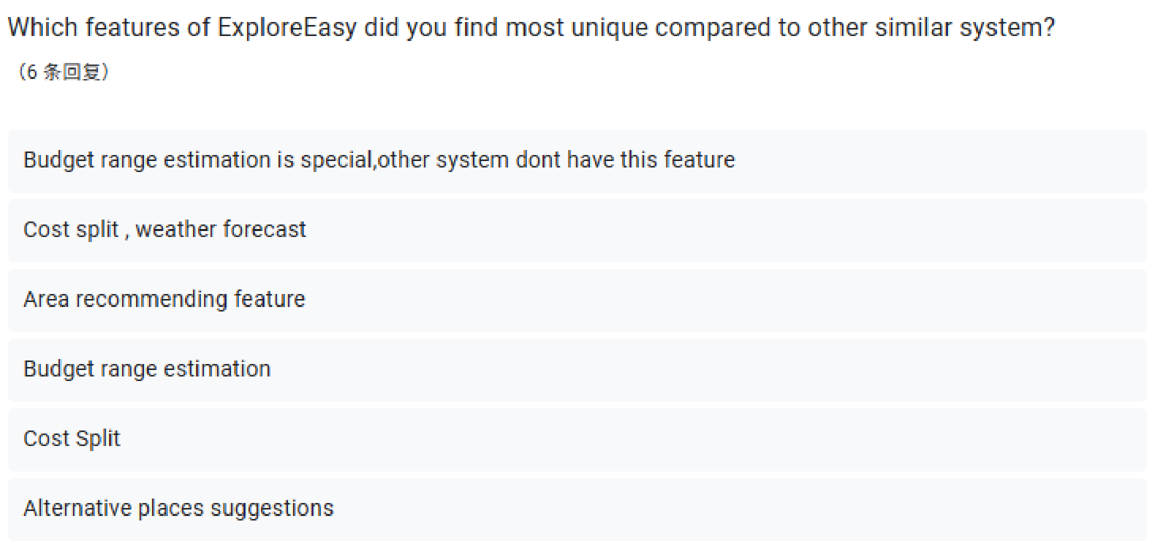


Figure 4.2.19 Overall user satisfaction with ExploreEasy

This chart shows that half of the respondents (50%) expressed the highest level of satisfaction with ExploreEasy, 33.3% rated their satisfaction at level 4, while 16.7% gave a lower score of 2. The majority of users (83.3%) therefore indicated positive experiences, though the small percentage of lower satisfaction suggests there are opportunities to further refine certain features, particularly in accommodation and budget management, to achieve even higher overall approval.


Which features of ExploreEasy did you find most unique compared to other similar system?
（6 条回复）

Budget range estimation is special,other system dont have this feature

Cost split , weather forecast

Area recommending feature

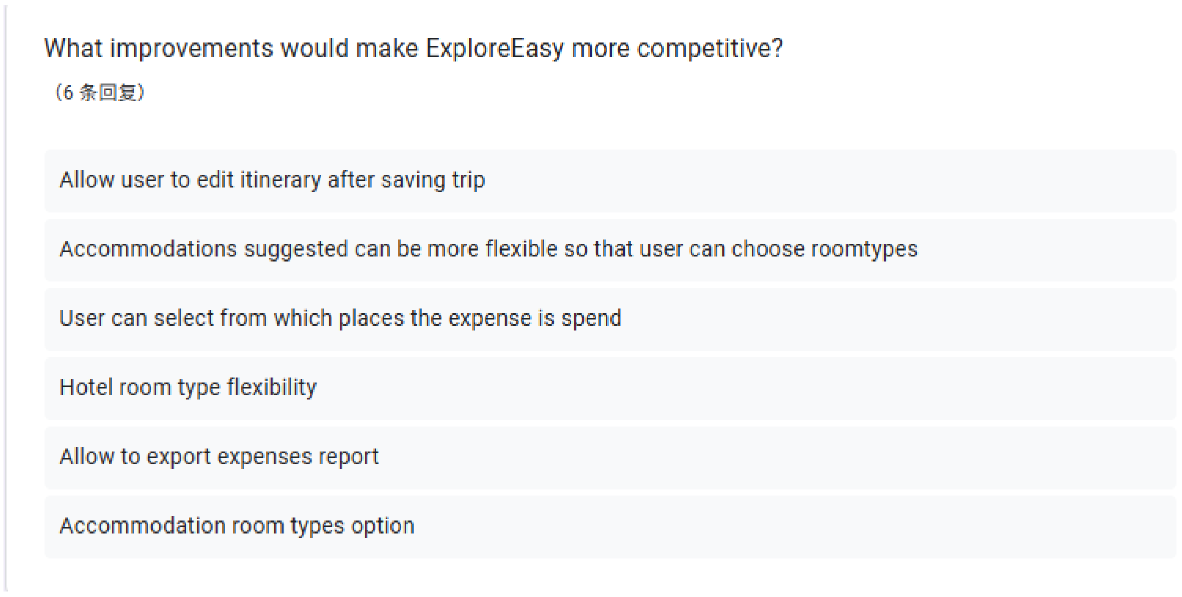
Budget range estimation

Cost Split

Alternative places suggestions

Figure 4.2.20 Respondents' feedback on unique features of ExploreEasy compared to other systems

The responses show that participants found several ExploreEasy features unique compared to other systems such as TripAdvisor or Wanderlog. The most frequently mentioned features include budget range estimation, cost splitting, area recommendation, weather forecast integration, and alternative place suggestions. Notably, budget range estimation was highlighted as a particularly distinctive feature, as respondents noted that competing systems do not offer this functionality. This indicates that ExploreEasy has successfully introduced innovative elements that differentiate it from existing travel planning applications.


What improvements would make ExploreEasy more competitive?
（6 条回复）

Allow user to edit itinerary after saving trip

Accommodations suggested can be more flexible so that user can choose roomtypes

User can select from which places the expense is spend

Hotel room type flexibility

Allow to export expenses report

Accommodation room types option

Figure 4.2.21 Suggested improvements to enhance ExploreEasy's competitiveness

This chart shows that respondents suggested several enhancements to make ExploreEasy more competitive in the travel planning market. The most frequently mentioned improvement was providing flexibility in accommodation selection, such as allowing users to choose room types, which was highlighted in multiple responses. Other suggestions include enabling users to edit itineraries even after saving a trip, and giving users control over selecting which places expenses are recorded from. Additionally, participants recommended adding the ability to export expense reports, which would improve financial tracking and usability. These suggestions emphasize the importance of customization, flexibility, and post-trip editing options, which can significantly increase user satisfaction and make ExploreEasy stand out further compared to existing competitors.

4.3 Results and Benchmark

The results of the survey provide valuable insights into how ExploreEasy performs compared to existing travel planning applications such as **TripAdvisor** and **Wanderlog**. The benchmarking analysis focused on several critical aspects: route planning, accommodation suggestions, budget management, and place recommendations.

Route Planning Efficiency (vs TripAdvisor)

Survey findings show that 83.3% of respondents rated ExploreEasy's route planning efficiency at level 4 and 16.7% at level 5 when compared to TripAdvisor. None of the respondents gave low ratings, which indicates that ExploreEasy performs strongly in this area. This suggests that the system's automated route optimization is competitive with, and in some cases superior to, TripAdvisor's capabilities, especially by minimizing travel time and distance.

Accommodation Suggestions (vs TripAdvisor)

In terms of accommodation suggestions, 50% of respondents gave ExploreEasy the highest rating, 33.3% rated it at level 4, but 16.7% provided a low rating. This indicates that while the majority of users were satisfied, ExploreEasy's accommodation recommendation feature is not yet consistently outperforming TripAdvisor. TripAdvisor remains more established in this domain, likely due to its wider hotel database and user reviews, which add credibility and reliability to its suggestions.

Budget Management (vs Wanderlog)

When benchmarked against Wanderlog, ExploreEasy's budget management received mixed results. While 50% of respondents rated the feature at the highest level and 16.7% at level 4, 33.3% gave a lower rating of 2. This shows that although many users appreciated ExploreEasy's real-time budget alerts and estimation functions, Wanderlog is still seen as stronger in providing advanced and reliable budgeting tools. This highlights an area for further refinement in ExploreEasy to meet user expectations.

Place Recommendations (vs Wanderlog)

ExploreEasy performed well in place recommendations compared to Wanderlog, with 50% of respondents rating it at level 4 and the remaining 50% at level 5. No negative ratings were received, showing that ExploreEasy's hybrid recommendation system is highly effective in matching users' travel interests with relevant destinations. This demonstrates that the system's combination of content-based and weather-aware filtering is a strong differentiator.

Unique Features of ExploreEasy

Survey responses also revealed that users found several features of ExploreEasy unique compared to other systems, including budget range estimation, cost splitting, weather-aware suggestions, area recommendations, and alternative place substitutions. These features were identified as key strengths that differentiate ExploreEasy from competitors.

Suggested Improvements

Participants also highlighted several areas for improvement, including:

- Greater flexibility in accommodation selection (e.g., ability to choose specific room types).
- The option to edit itineraries after saving a trip.
- Enhanced expense management, including selecting expense sources and exporting reports.
- Improved accuracy in budget range estimation to reduce uncertainty.

Overall, ExploreEasy demonstrates strong competitiveness in route planning, place recommendations, and unique features such as weather-aware suggestions and cost splitting. However, TripAdvisor still outperforms ExploreEasy in accommodation reliability due to its extensive database and user reviews, while Wanderlog maintains an advantage in advanced budget management.

Therefore, ExploreEasy can be considered **better suited for users who value convenience, collaborative planning, and unique automation features**, whereas **TripAdvisor and Wanderlog remain stronger in specific traditional areas such as accommodation credibility and budget robustness**. To become more competitive, ExploreEasy should focus on improving hotel recommendation accuracy, offering greater customization options, and strengthening its budget management functions.

4.4 Objectives Evaluation

Objective 1: AI-Driven Trip Planning and Hybrid Recommendation with Weather Forecasting

The first objective was to develop an AI-driven trip planning system that integrates **Collaborative Filtering (CF)** and **Content-Based Filtering (CBF)** to provide personalized recommendations, with **weather forecasting** to adapt itineraries. The survey results indicate that this objective has been successfully met. Respondents rated the **relevance of recommended areas** highly, with 83.3% giving the maximum rating, showing strong alignment between system recommendations and user interests. Additionally, the **weather-aware feature** received unanimous approval (100%), as users confirmed that avoiding outdoor activities during rain improved their planning experience. Furthermore, **route optimization** was also rated positively, with 66.7% giving it the highest score. These results confirm that the hybrid recommendation system, weather integration, and TSP-based optimization improved efficiency, adaptability, and personalization, thereby meeting user satisfaction.

Objective 2: Budget Estimation and Management with Expense Control
The second objective focused on implementing a comprehensive budget estimation and management module, incorporating Jaccard Similarity, Weighted Averaging, and real-time alerts. The survey findings demonstrate that this objective was largely achieved but with some mixed results. In terms of budget estimation accuracy, 66.7% rated it as accurate and 33.3% as very accurate in one survey set, but another chart showed a split where 50% responded "Yes," 33.3% "Maybe," and 16.7% "No." This indicates that while many users trusted the budget estimation, some expressed uncertainty about its precision. On the other hand, budget alerts received unanimous positive feedback (100%), confirming that real-time monitoring significantly helped users maintain financial control. The cost splitting feature also achieved full approval (100%), showing its usefulness in group travel scenarios. When benchmarked against Wanderlog, ExploreEasy's budget management received both high (50% gave top score) and lower ratings (33.3% scored 2), suggesting that while ExploreEasy introduced unique features like range estimation and alerts, Wanderlog remains stronger in terms of robustness. Overall, this objective can be considered partially achieved with strong user satisfaction in cost control but room for improvement in estimation precision.

Objective 3: Flexibility through Similar-Place Substitution

The third objective aimed to provide **similar-place substitution** to increase flexibility, supported by geographic filtering and quality thresholds. The survey results confirm that this objective was **fully achieved**. All respondents (100%) agreed that the substitution feature improved flexibility in their trip planning. Furthermore, satisfaction with the **quality of alternate suggestions** was very high, with 66.7% rating it at level 4 and 33.3% at level 5. This indicates that the algorithm for alternative recommendations successfully met expectations by ensuring substitutes were both relevant and practical within the itinerary. As a result, users gained confidence that their plans could adapt to unexpected circumstances while still maintaining quality and enjoyment.

Overall, the evaluation of project objectives against survey results shows that **Objectives 1 and 3 were successfully met with high levels of user satisfaction**, while **Objective 2 was partially met** due to mixed perceptions of budget estimation accuracy despite strong approval for cost control features. Compared with benchmarking systems, ExploreEasy excels in personalization, weather-aware planning, and itinerary flexibility, while competitors like TripAdvisor and Wanderlog remain stronger in accommodation credibility and budget robustness. To further improve, ExploreEasy should refine its budget estimation model, increase accommodation selection flexibility (e.g., room type options), and allow itinerary editing after saving, as suggested by users.

4.5 Concluding Remark

The overall results of the survey and benchmarking analysis demonstrate that ExploreEasy has achieved most of its intended objectives and provides several innovative features that differentiate it from existing travel planning applications. The system performed particularly well in delivering **personalized recommendations, weather-aware itinerary planning, route optimization, cost splitting, and substitution flexibility**, all of which received consistently positive feedback from users. These findings highlight ExploreEasy’s strength in enhancing convenience, adaptability, and collaboration for travelers.

However, the evaluation also revealed areas requiring further refinement. While users were generally satisfied with the **budget estimation and accommodation suggestions**, some respondents expressed uncertainty about the precision of budget ranges and the reliability of hotel recommendations compared to established platforms such as Wanderlog and TripAdvisor. Open-ended feedback also pointed to desired improvements such as **greater flexibility in accommodation selection, the ability to edit itineraries after saving, and options to export expense reports**, which would further enhance competitiveness and user satisfaction.

In conclusion, ExploreEasy can be considered a promising system that offers **unique, user-centered features not commonly found in competing platforms**, while still requiring incremental improvements in accuracy, flexibility, and financial robustness to reach its full potential. These results confirm that the system has strong potential to complement and even surpass existing travel planning applications in certain aspects, while future enhancements will further establish its position as a comprehensive AI-driven travel management tool.

Chapter 5
Conclusion

This project, *ExploreEasy: Smart and All-in-One Trip Management Application*, set out to address the challenges faced by modern travellers in creating cost-effective, flexible, and personalised travel experiences. The system was designed to consolidate planning, budgeting, and adaptability into a single intelligent platform.

The objectives established at the beginning of this project were as follows:

1. **To develop an AI-driven trip planning and recommendation system using a hybrid recommendation algorithm with integrated real-time and extended weather forecasting for efficient, adaptive, and personalised travel planning.** This objective was successfully met. The hybrid recommendation engine combining Collaborative Filtering (CF) and Content-Based Filtering (CBF) was implemented to provide customised suggestions. Route optimisation via the Travelling Salesman Problem (TSP) and weather-aware itinerary adjustments ensured that travel plans were both efficient and adaptive to real-time conditions.
2. **To implement a comprehensive budget estimation and management module using Jaccard Similarity and Weighted Averaging for accurate budget guidance and effective travel expense control.**
   This objective was also achieved. The budget module provided reliable cost estimation by applying similarity-based filtering techniques, inflation adjustments, and percentile trimming to ensure accuracy. Additionally, features such as category-based allocation, cost splitting, and real-time budget alerts were incorporated, enabling effective financial management during trips.
3. **To enhance flexibility in itinerary planning by implementing a reliable similar-place substitution feature using Geographic Filtering and Quality Thresholds.** This objective was accomplished. The substitution module allowed users to replace unavailable or unsuitable places with nearby and quality-assured alternatives, preserving itinerary coherence and improving user satisfaction.

In summary, all project objectives have been fulfilled. The ExploreEasy system successfully integrates intelligent trip planning, budget management, and flexible itinerary adjustment into a cohesive platform. User evaluation confirmed improvements in personalisation, adaptability, and financial confidence, positioning ExploreEasy as a more advanced and user-centric solution compared to existing travel management systems.

While ExploreEasy has successfully achieved its objectives, several enhancements could be pursued in future iterations. First, extending weather forecasting through seasonal trend analysis and machine learning would enable more accurate planning for long-duration trips. Second, integration with real-time booking APIs for hotels and transport could allow seamless reservations, dynamic pricing updates, and cancellation handling directly within the platform. Third, adding an expense export feature that allows users to generate and download detailed reports in Excel format would improve transparency, support post-trip financial reviews, and make it easier to share costs among group travelers. Additionally, enabling offline functionality with cached maps, itineraries, and expense records would improve usability in low-connectivity regions, while edge AI could support lightweight recommendations. Finally, incorporating gamification and collaborative planning features would increase engagement and make group travel coordination more interactive and enjoyable.

REFERENCES

[1] C. Andri, M. Hazim Alkawaz, and A. Bibo Sallow, “Adoption of Mobile Augmented Reality as a Campus Tour Application,” International Journal of Engineering & Technology, vol. 7, no. 4.11, p. 64, Oct. 2018, doi: https://doi.org/10.14419/ijet.v7i4.11.20689.

[2] Kim, “How to Deal With Shared Finances On a Group Trip – My Global Ways,” My Global Ways, Aug. 05, 2020. https://myglobalways.com/index.php/2020/08/05/how-to-deal with-finances-on-a- group-trip/ (accessed Sep. 11, 2024).

[3] U. P. Singh, M. K. P. Singh, M. Sharma, and A. Sharma, "Spending Tracker: A Smart Approach to Track Daily Expense," Turkish Journal of Computer and Mathematics Education, vol. 12, no. 6, pp. 5095–5103, 2021. [Online]. Available: https://doi.org/10.17762/turcomat.v12i6.8759

[4] H.-T. Chang, Y.-M. Chang, and M.-T. Tsai, “ATIPS: Automatic Travel Itinerary Planning System for Domestic Areas,” *Computational Intelligence and Neuroscience*, vol. 2016, pp. 1–13, 2016.

[5] G. Adomavicius and A. Tuzhilin, “Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 6, pp. 734-749, 2005.

[6] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, “Item-based collaborative filtering recommendation algorithms,” in *Proceedings of the 10th International Conference on World Wide Web*, 2001, pp. 285-295.

[7] Y. Koren, R. Bell, and C. Volinsky, “Matrix factorization techniques for recommender systems,” *Computer*, vol. 42, no. 8, pp. 30-37, 2009.

[8] X. He et al., “Neural collaborative filtering,” in *Proceedings of the 26th International Conference on World Wide Web*, 2017, pp. 173-182.

[9] P. Melville, R. J. Mooney, and R. Nagarajan, “Content-boosted collaborative filtering for improved recommendations,” in *Proceedings of the 18th National Conference on Artificial Intelligence*, 2002, pp. 187-192.

[10] M. J. Pazzani and D. Billsus, “Content-based recommendation systems,” in *The Adaptive Web*, P. Brusilovsky, A. Kobsa, and W. Nejdl, Eds. Berlin, Heidelberg: Springer, 2007, pp. 325-341.

[11] A. Brown and B. Lee, "The Jaccard Index: Applications in Data Analysis," *J. Appl. Stat.*, vol. 47, no. 3, pp. 512–525, 2020.

[12] D. Chen and F. Wang, "Comparative Study of Similarity Measures for Sparse Data," in *Proc. Int. Conf. Data Mining*, pp. 124–138, 2021.

[13] G. Davis and H. Thompson, "The Role of Similarity Metrics in Information Retrieval," *IEEE Trans. Knowl. Data Eng.*, vol. 31, no. 8, pp. 1540–1551, 2019.

[14] A. Ganti, "Weighted Average Definition," *Investopedia*, Oct. 08, 2024. https://www.investopedia.com/terms/w/weightedaverage.asp
[15] J. Kirchner, "Data Analysis Toolkit #12: Weighted averages and their uncertainties," 2006. [Online]. Available: https://seismo.berkeley.edu/~kirchner/Toolkits/Toolkit_12.pdf.

[16] Y. Yu, "Analysis and Study on Intelligent Tourism Route Planning Scheme Based on Weighted Mining Algorithm," *Scientific Programming*, vol. 2022, pp. 1–9, Jul. 2022, doi: https://doi.org/10.1155/2022/5495822.

[17] A. Toloie-Eshlaghy, M. Homayonfar, M. Aghaziarati, and P. Arbabiun, "A Subjective Weighting Method Based on Group Decision Making For Ranking and Measuring Criteria Values," *Australian Journal of Basic and Applied Sciences*, vol. 5, no. 12, pp. 2034–2040, 2011, Available: https://www.ajbasweb.com/old/ajbas/2011/December-2011/2034-2040.pdf

[18] M. Künnemann and B. Manthey, "On the Smoothed Approximation Ratio of the 2-Opt Heuristic for the TSP." [Online]. Available: https://ris.utwente.nl/ws/portalfiles/portal/5366028/paper_6.pdf

[19] Gokul Krishna M, M. Haseeb, Mohammed Siyad B, P.A Mohamed Zameel, and S Vyshnav Raj, "Budget and Experience Based Travel Planner Using Collaborative Filtering," Budget and Experience Based Travel Planner Using Collaborative Filtering, Jan. 2021, doi: https://doi.org/10.1109/odicon50556.2021.9428978.

[20] D. Glez-Peña, A. Lourenço, H. López-Fernández, M. Reboiro-Jato, and F. Fdez-Riverola, "Web scraping technologies in an API world," Briefings in Bioinformatics, vol. 15, no. 5, pp. 788–797, Apr. 2013, doi: https://doi.org/10.1093/bib/bbt026.

[21] Abhinav Ajitsaria, "Build a Recommendation Engine With Collaborative Filtering," Realpython.com, Jul. 10, 2019. https://realpython.com/build- recommendation-enginecollaborative-filtering/

[22] Thom Snaphaan, Wim Hardyns, Lieven J.R. Pauwels, and K. Bowers, "Rating places and crime prevention: Exploring user-generated ratings to assess place management," Computers Environment and Urban Systems, vol. 109, pp. 102088– 102088, Apr. 2024, doi: https://doi.org/10.1016/j.compenvurbsys.2024.102088.

[23] Y. Afoudi, M. Lazaar, and M. Al Achhab, "Hybrid recommendation system combined content-based filtering and collaborative prediction using artificial neural network," Simulation Modelling Practice and Theory, vol. 113, p. 102375, Jul. 2021, doi: https://doi.org/10.1016/j.simpat.2021.102375.

[24] W. Loh, Leong, U. Tunku, and A. Rahman, "Itinerary Planner for Tourism Mobile Application with Smart Trip Generation and Social Platform A REPORT SUBMITTED TO," 2022. Accessed: Jan. 18, 2024. [Online]. Available: http://eprints.utar.edu.my/4706/1/fyp_CS_2022_LWL.pdf

[25] “Wanderlog: travel itinerary, vacation & road trip planner,” Wanderlog. https://wanderlog.com/

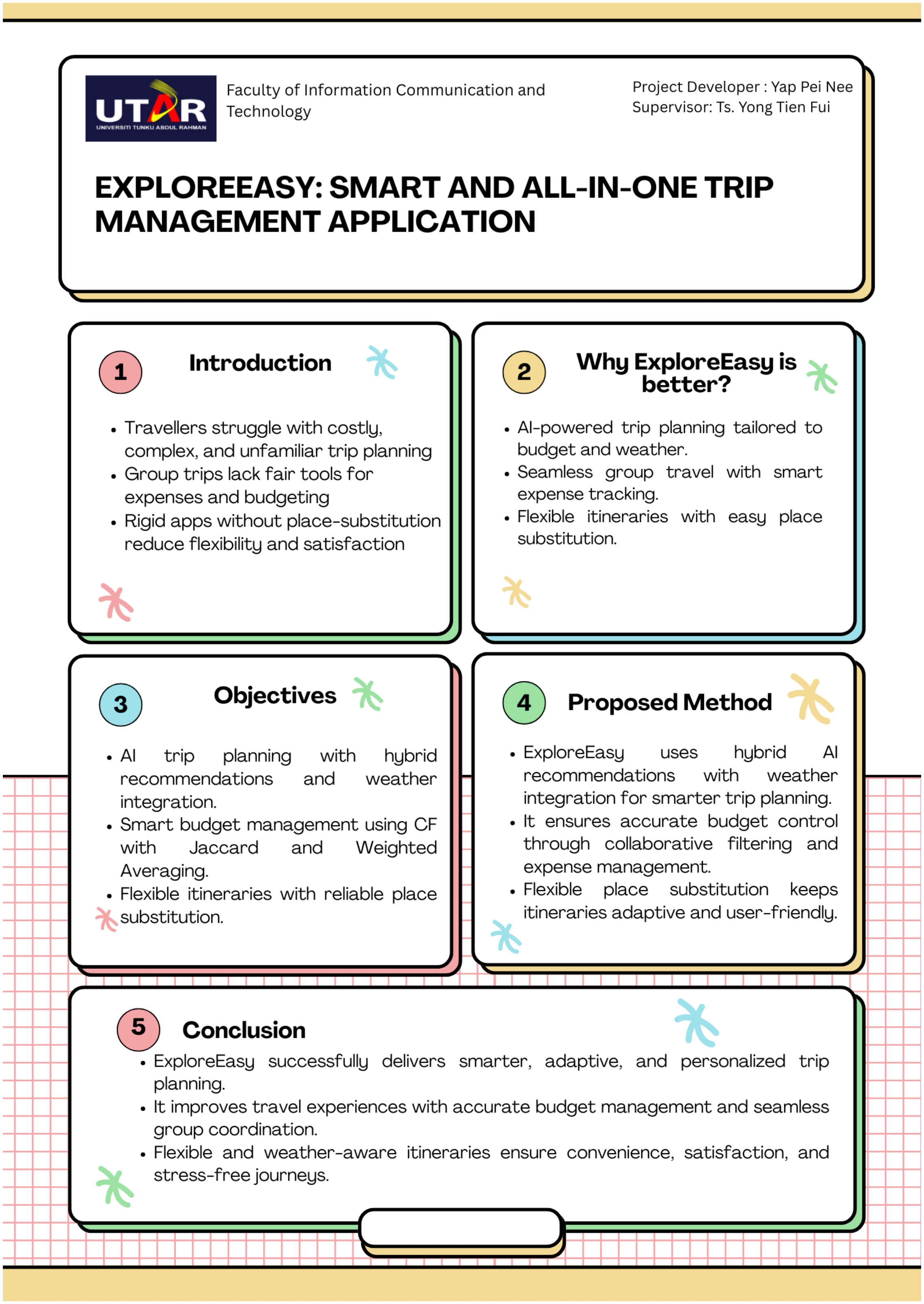
[26] Wanderlog, “Wanderlog Itinerary & Road Trip Planner,” Wanderlog, 2024. https://wanderlog.com/trip-plan-assistant (accessed Sep. 11, 2024).

[27] C. Alaimo, J. Kallinikos, and E. Valderrama-Venegas, “Platform Evolution: A Study of TripAdvisor,” scholarspace.manoa.hawaii.edu, Jan. 07, 2020. http://hdl.handle.net/10125/64414

[28] M. Nilashi, O. Ibrahim, E. Yadegaridehkordi, S. Samad, E. Akbari, and A. Alizadeh, “Travelers decision making using online review in social network sites: A case on TripAdvisor,” Journal of Computational Science, vol. 28, pp. 168–179, Sep. 2018, doi: https://doi.org/10.1016/j.jocs.2018.09.006.

[29] “The Power of Reviews How Tripadvisor Reviews Lead to Bookings and Better Travel Experiences.” Available: https://www.tripadvisor.com/powerofreviews.pdf

POSTER



Faculty of Information Communication and Technology

Project Developer : Yap Pei Nee
Supervisor: Ts. Yong Tien Fui

EXPLOREEASY: SMART AND ALL-IN-ONE TRIP MANAGEMENT APPLICATION

1 Introduction

- Travellers struggle with costly, complex, and unfamiliar trip planning
- Group trips lack fair tools for expenses and budgeting
- Rigid apps without place-substitution reduce flexibility and satisfaction

2 Why ExploreEasy is better?

- AI-powered trip planning tailored to budget and weather.
- Seamless group travel with smart expense tracking.
- Flexible itineraries with easy place substitution.

3 Objectives

- AI trip planning with hybrid recommendations and weather integration.
- Smart budget management using CF with Jaccard and Weighted Averaging.
- Flexible itineraries with reliable place substitution.

4 Proposed Method

- ExploreEasy uses hybrid AI recommendations with weather integration for smarter trip planning.
- It ensures accurate budget control through collaborative filtering and expense management.
- Flexible place substitution keeps itineraries adaptive and user-friendly.

5 Conclusion

- ExploreEasy successfully delivers smarter, adaptive, and personalized trip planning.
- It improves travel experiences with accurate budget management and seamless group coordination.
- Flexible and weather-aware itineraries ensure convenience, satisfaction, and stress-free journeys.